BMLIP-5SSD0

Bayesian Machine Learning and Information Processing

Lecture Notes

Bert de Vries

Wouter Kouw (Flux 7.060)

Magnus T. Koudahl (Flux 7.060)

5SSD0 Course Syllabus

Learning Goals

- This course provides an introduction to Bayesian machine learning and information processing systems. The Bayesian approach affords a unified and consistent treatment of many useful information processing systems.
- Upon successful completion of the course, students should be able to:
  - understand the essence of the Bayesian approach to information processing.
  - specify a solution to an information processing problem as a Bayesian inference task on a probabilistic model.
  - design a probabilistic model by a specifying a likelihood function and prior distribution;
  - Code the solution in a probabilistic programming package.
  - execute the Bayesian inference task either analytically or approximately.
  - evaluate the resulting solution by examination of Bayesian evidence.
  - be aware of the properties of commonly used probability distribitions such as the Gaussian, Gamma and multinomial distribution; models such as hidden Markov models and Gaussian mixture models; and inference methods such as the Laplace approximation, variational Bayes and message passing in a factor graph.

Entrance Requirements (pre-knowledge)

- Undergraduate courses in Linear Algebra and Probability Theory (or Statistics).
- Some scientific programming experience, eg in MATLAB or Python. In this class, we use the Julia programming language, which has a similar syntax to MATLAB, but is (close to) as fast as C.

Instructors

- Bert de Vries, rm. FLUX-7.101, overall responsible lecturer
- Wouter Kouw, rm. FLUX-7.060, responsible for Probabilistic Programming lessons
- Magnus Koudahl, rm. FLUX-7.060, teaching assistant

Important Links

- Please bookmark the following three websites:
  1. The course homepage http://bmlip.nl (or find the site by navigating to the teaching tab at http://biaslab.org) contains links to all materials such as lecture notes and video lectures.
  2. The Piazza course site will be used for Q&A and communication.
  3. The Canvas course site will be sparingly used for communication (mostly by ESA staff)

Materials

- All materials can be accessed from the course homepage http://bmlip.nl.
- Materials consist of the following resources:
  - Lecture notes (mandatory)
  - Probabilistic Programming (PP) notes (mandatory)
  - optional materials to help understand the lecture and PP notes
    - video guides to the lecture notes
    - live class recording (of the 2020 course) and (unrecorded) live sessions at regular class hours for this term
    - exercises
    - Q&A at Piazza
    - practice exams
- Source materials are available at github repo at https://github.com/bertdv/BMLIP. You do not need to bother with this site. If you spot an error in the materials, please raise the issue at Piazza.

Lecture and PP notes and Video Guides

- The lecture and PP notes contain the mandatory materials. Some lecture notes are extended by a reading assignment, see the first cell in the lecture notes. These reading assignment are also part of the mandatory materials.
- Slides that are not required for the exam are moved to the end of the notes and preceded by an OPTIONAL SLIDES header.
- The accompanying Video Guides aim to cover just the main points or (expected) sticky issues in a lecture.

Study Guide

- Here's our recommendation on how to study for this class. Before each lecture:
  - First watch the video guide for that lecture
  - Then study the lecture notes
  - Then, optionally, watch the live class recording from the previous (2020/21) edition
  - Then try to make the exercises for that class.
    - When you do the exercises, feel free to make use of Sam Roweis' cheat sheets for Matrix identities and Gaussian identities. Also accessible from course homepage.
  - Optionally, come to the live class at regular class hours to discuss remaining issues in person.
- Aside from the regular classes, you are encouraged to disscuss any issues in Piazza. Your questions will be answered at the Piazza site by fellow students and accorded (or dorrected, amended) by the teaching staff.

Piazza (Q&A)

- We will be using Piazza for class discussion. The system is highly catered to getting you help fast and efficiently from both classmates and the teaching staff.
- Sign up for Piazza today if you have not done so. And install the Piazza app on your phone!
- The quicker you begin asking questions on Piazza (rather than via emails), the quicker you'll benefit from the collective knowledge of your classmates and instructors. We encourage you to ask questions when you're struggling to understand a concept—you can even do so anonymously.
- We will also disseminate news and announcements via Piazza.
- Unless it is a personal issue, pose your course-related questions at Piazza (in the right folder).
- Please contribute to the class by answering questions at Piazza.
  - If so desired, you can contribute anonymously.
  - Answering technical questions at Piazza is a great way to learn. If you really want to understand a topic, you should try to explain it to others.
  - Every question has just a single students' answer that students can edit collectively (and a single instructors' answer for instructors).
- You can use LaTeX in Piazza for math (and please do so!).
- Piazza has a great `search` feature. Use search before putting in new questions.

Live Class Sessions

- We will hold live class sessions at the regular class hours.
- The live classes are an opportunity to speak with the teaching staff directly about any issues.
- The live sessions will probably be short: as much as possible, I'd like to address technical questions and issues through Piazza so they are more easily accessible and searchable afterwards.

Exam Guide

- There will be a written exam in multiple-choice format.
- You are not allowed to use books nor bring printed or handwritten formula sheets to the exam. Difficult-to-remember formulas are supplied at the exam sheet.
- No smartphones at the exam.
- The tested material consists of the lecture + PP notes (+ reading assignments as assigned in the first cell/slide of each lecture notebook).
- The class homepage contains two representative practice exams from previous terms.

OPTIONAL SLIDES

Machine Learning Overview

Preliminaries

- Goal
  - Top-level overview of machine learning
- Materials
  - Mandatory
    - this notebook
  - Optional
    - pre-recorded video guide and live class (2020)
    - Study Bishop pp. 1-4

What is Machine Learning?

- Machine Learning relates to building models from data and using these models in applications.

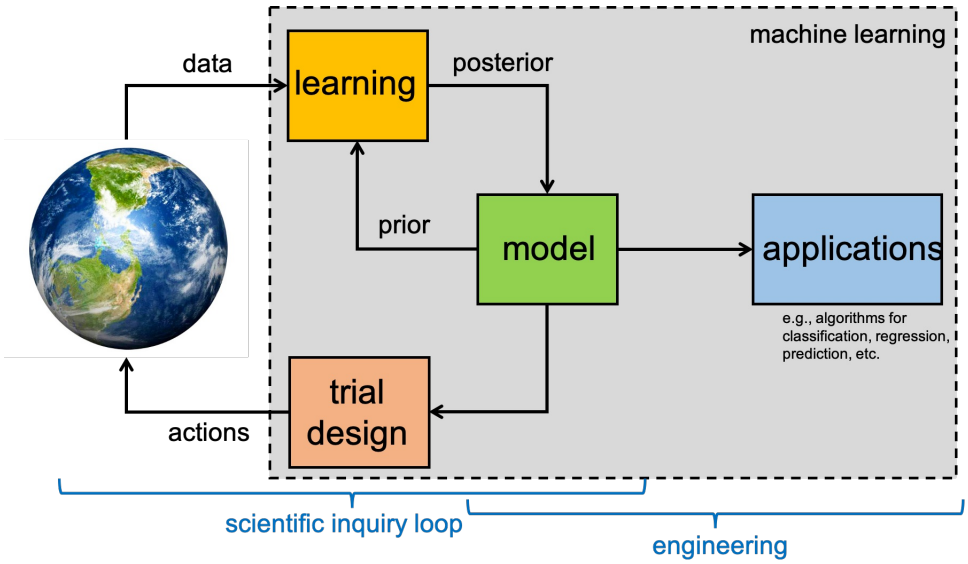
- Problem: Suppose we want to develop an algorithm for a complex process about which we have little knowledge (so hand-programming is not possible).

- Solution: Get the computer to develop the algorithm by itself by showing it examples of the behavior that we want.

- Practically, we choose a library of models, and write a program that picks a model and tunes it to fit the data.

- This field is known in various scientific communities with slight variations under different names such as machine learning, statistical inference, system identification, data mining, source coding, data compression, data science, etc.

Machine Learning and the Scientific Inquiry Loop



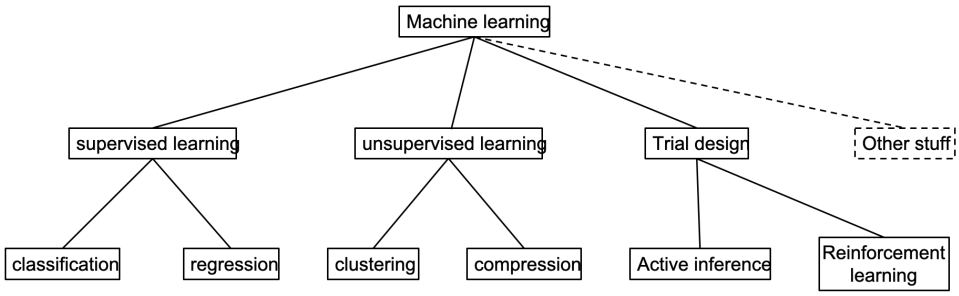
- Machine learning technology uses the scientific inquiry loop to develop models and use these models in applications.

Machine Learning is Difficult

- Modeling (Learning) Problems
  - Is there any regularity in the data anyway?
  - What is our prior knowledge and how to express it mathematically?
  - How to pick the model library?
  - How to tune the models to the data?
  - How to measure the generalization performance?

- Quality of Observed Data
  - Not enough data
  - Too much data?
  - Available data may be messy (measurement noise, missing data points, outliers)

A Machine Learning Taxonomy



- Supervised Learning: Given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  - Examples: classification, regression, time series prediction

- Unsupervised Learning: (a.k.a. density estimation). Given only inputs, automatically discover representations, features, structure, etc.
  - Examples: clustering, outlier detection, compression

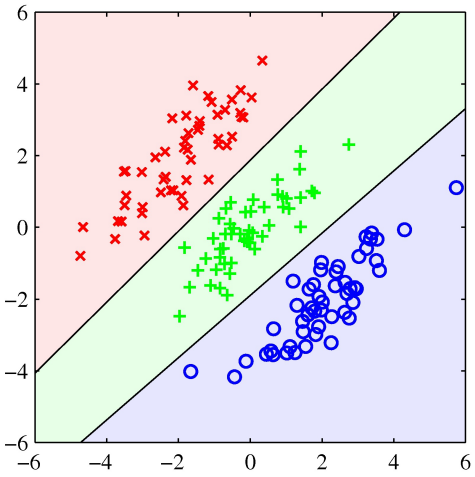
- Trial Design: (a.k.a. experimental design, active learning). Learn to make actions that optimize some performance criterion about the expected future.
  - Examples: playing games like chess, self-driving cars, robotics.
  - Two major approaches include reinforcement learning and active inference
    - Reinforcement Learning: Given an observed sequence of input signals and (occasionally observed) rewards for those inputs, learn to select actions that maximize expected future rewards.
    - Active inference: Given an observed sequence of input signals and a prior probability distribution about future observations, learn to select actions that minimize expected prediction errors (i.e., minimize actual minus predicted sensation).

- Other stuff, like preference learning, learning to rank, etc., can often be (re-)formulated as special cases of either a supervised, unsupervised or trial design problem.

Supervised Learning

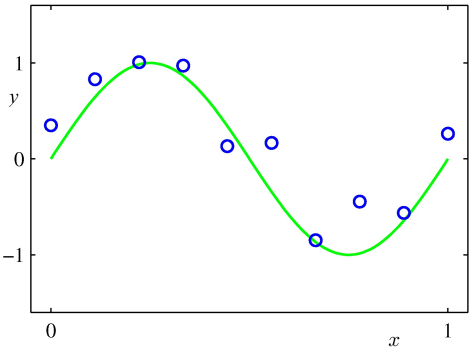
- Given observations of desired input-output behavior $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ (with x_i inputs and y_i outputs), the goal is to estimate the conditional distribution $p(y \mid x)$ (i.e., how does y depend on x?).

Classification



- The target variable y is a discrete-valued vector representing class labels
- The special case $y \in \{\text{true}, \text{false}\}$ is called detection.

Regression

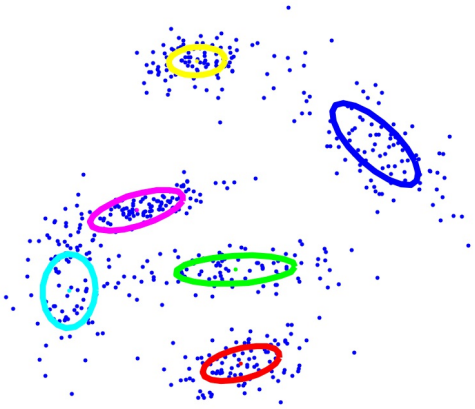


- Same problem statement as classification but now the target variable is a real-valued vector.
- Regression is sometimes called curve fitting.

Unsupervised Learning

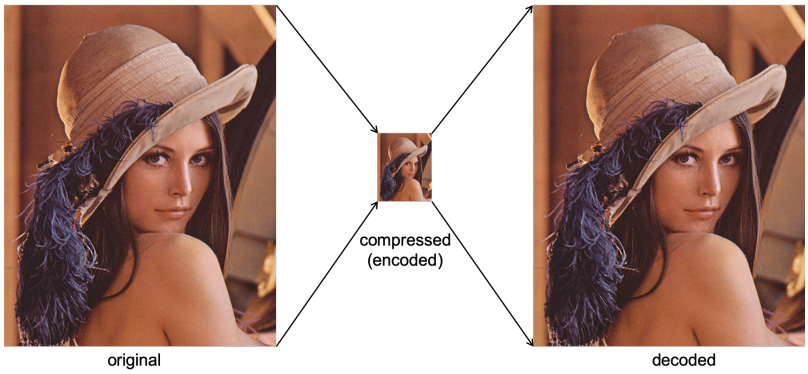
Given data $D = \{x_1, \ldots, x_N\}$, model the (unconditional) probability distribution $p(x)$ (a.k.a. density estimation). The two primary applications are clustering and compression (a.k.a. dimensionality reduction).

Clustering



- Group data into clusters such that all data points in a cluster have similar properties.
- Clustering can be interpreted as "unsupervised classification".

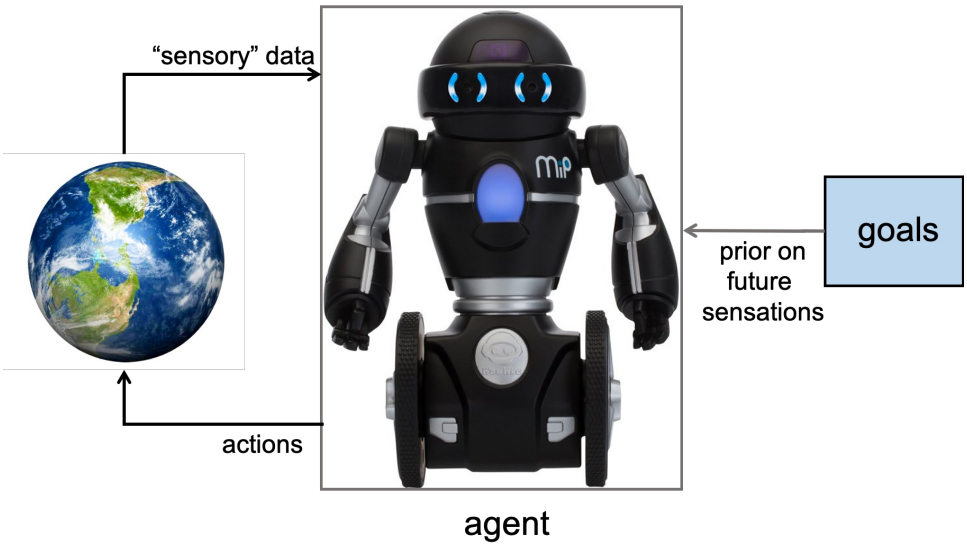
Compression / dimensionality reduction



- Output from coder is much smaller in size than original, but if coded signal if further processed by a decoder, then the result is very close (or exactly equal) to the original.
- Usually, the compressed image comprises continuously valued variables. In that case, compression can be interpreted as "unsupervised regression".

Trial Design

Given the state of the world (obtained from sensory data), the agent must learn to produce actions that optimize some performance criterion about expected future.



- In this course, we focus on the active inference approach; see Intelligent Agent lesson.

Some Machine Learning Applications

- computer speech recognition, speaker recognition
- face recognition, iris identification
- printed and handwritten text parsing
- financial prediction, outlier detection (credit-card fraud)
- user preference modeling (amazon); modeling of human perception
- modeling of the web (google)
- machine translation
- medical expert systems for disease diagnosis (e.g., mammogram)
- strategic games (chess, go, backgammon), self-driving cars
- In summary, any 'knowledge-poor' but 'data-rich' problem

Probability Theory Review

Preliminaries

- Goal
  - Review of probability theory as a theory for rational/logical reasoning with uncertainties (i.e., a Bayesian interpretation)
- Materials
  - Mandatory
    - These lecture notes
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.7-26 (sections 2.1 through 2.5), on deriving probability theory. You may skip section 2.3.4: Cox's proof (pp.15-18).
  - Optional
    - the pre-recorded video guide and live class of 2020
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.7-56 (ch.2: probability)
      - Great introduction to probability theory, in particular w.r.t. its correct interpretation as a state-of-knowledge.
      - Absolutely worth your time to read the whole chapter!
    - Edwin Jaynes - 2003 - Probability Theory -- The Logic of Science.
      - Brilliant book on Bayesian view on probability theory. Just for fun, scan the annotated bibliography and references.
    - Bishop pp. 12-24

Example Problem: Disease Diagnosis

- Problem: Given a disease with prevalence of 1% and a test procedure with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85% , what is the chance that somebody who tests positive actually has the disease?

- Solution: Use probabilistic inference, to be discussed in this lecture.

The Design of Probability Theory

- Define an event (or "proposition") A as a statement, whose truth can be contemplated by a person, e.g.,

$$A = \text{'there is life on Mars'}$$

- If we assume the fact

  $$I = \text{'All known life forms require water'}$$

  and a new piece of information

  $$x = \text{'There is water on Mars'}$$

  becomes available, how should our degree of belief in event A be affected (if we were rational)?

- Richard T. Cox, 1946 developed a calculus for rational reasoning about how to represent and update the degree of beliefs about the truth value of events when faced with new information.

- In developing this calculus, only some very agreeable assumptions were made, e.g.,
  - (Transitivity). If the belief in A is greater than the belief in B, and the belief in B is greater than the belief in C, then the belief in A must be greater than the belief in C.
  - (Consistency). If the belief in an event can be inferred in two different ways, then the two ways must agree on the resulting belief.

- This effort resulted in confirming that the sum and product rules of Probability Theory are the only proper rational way to process belief intensities.

- ⇒ Probability theory (PT) provides the theory of optimal processing of incomplete information (see Cox theorem, and Caticha, pp.7-24), and as such provides the (only) proper quantitative framework for drawing conclusions from a finite (read: incomplete) data set.

Why Probability Theory for Machine Learning?

- Machine learning concerns drawing conclusions about model parameter settings from (a finite set of) data and therefore PT provides the optimal calculus for machine learning.

- In general, nearly all interesting questions in machine learning can be stated in the following form (a conditional probability):

  $$p(\text{whatever-we-want-to-know} \mid \text{whatever-we-do-know})$$

where $p(a \mid b)$ means the probability that a is true, given that b is true.

- Examples
  - Predictions

$$p(\text{ future-observations } | \text{ past-observations })$$

  - Classify a received data point x

$$p(x\text{-belongs-to-class-}k \mid x)$$

  - Update a model based on a new observation

$$p(\text{ model-parameters } | \text{ new-observation, past-observations })$$

Frequentist vs. Bayesian Interpretation of Probabilities

- The interpretation of a probability as a degree-of-belief about the truth value of an event is also called the Bayesian interpretation.

- In the Bayesian interpretation, the probability is associated with a state-of-knowledge (usually held by a person).
  - For instance, in a coin tossing experiment, $p(\text{tail}) = 0.4$ should be interpreted as the belief that there is a 40% chance that **tail** comes up if the coin were tossed.
  - Under the Bayesian interpretation, PT calculus (sum and product rules) extends boolean logic to rational reasoning with uncertainty.

- The Bayesian interpretation contrasts with the frequentist interpretation of a probability as the relative frequency that an event would occur under repeated execution of an experiment.
  - For instance, if the experiment is tossing a coin, then $p(\text{tail}) = 0.4$ means that in the limit of a large number of coin tosses, 40% of outcomes turn up as **tail**.

- The Bayesian viewpoint is more generally applicable than the frequentist viewpoint, e.g., it is hard to apply the frequentist viewpoint to events like '**it will rain tomorrow**'.

- The Bayesian viewpoint is clearly favored in the machine learning community. (In this class, we also strongly favor the Bayesian interpretation).

Probability Theory Notation

events

- Define an event A as a statement, whose truth can be contemplated by a person, e.g.,

$$A = \text{'it will rain tomorrow'}$$

- We write the denial of A, i.e. the event not-A, as $\bar{A}$.

- Given two events A and B, we write the conjunction "$A \wedge B$" as "A, B" or "AB". The conjunction AB is true only if both A and B are true.

- We will write the disjunction "$A \vee B$" as "$A + B$", which is true if either A or B is true or both A and B are true.

- Note that, if X is a variable, then an assignment $X = x$ (with x a value, e.g., $X = 5$) can be interpreted as an event.

probabilities

- For any event A, with background knowledge I, the conditional probability of A given I, is written as

$$p(A \mid I).$$

- All probabilities are in principle conditional probabilities of the type $p(A \mid I)$, since there is always some background knowledge.

Unfortunately, PT notation is usually rather sloppy :(

- We often write $p(A)$ rather than $p(A \mid I)$ if the background knowledge I is assumed to be obviously present. E.g., $p(A)$ rather than $p(A \mid \text{the-sun-comes-up-tomorrow})$.

- (In the context of random variable assignments) we often write $p(x)$ rather than $p(X = x)$, assuming that the reader understands the context.

- In an apparent effort to further abuse notational conventions, $p(X)$ denotes the full distribution over random variable X, i.e., the distribution for all assignments for X.

- If X is a discretely valued variable, then $p(X = x)$ is a probability mass function (PMF) with $0 \leq p(X = x) \leq 1$ and normalization $\sum_x p(x) = 1$.

- If X is continuously valued, then $p(X = x)$ is a probability density function (PDF) with $p(X = x) \geq 0$ and normalization $\int_x p(x)\mathrm{d}x = 1$.

  - Note that if X is continuously valued, then the value of the PDF $p(x)$ is not necessarily ≤ 1. E.g., a uniform distribution on the continuous domain $[0, .5]$ has value $p(x) = 2$.

- Often, we do not bother to specify if $p(x)$ refers to a continuous or discrete variable.

Probability Theory Calculus

- Let $p(A|I)$ indicate the belief in event A, given that I is true.

- The following product and sum rules are also known as the axioms of probability theory, but as discussed above, under some mild assumptions, they can be derived as the unique rules for rational reasoning under uncertainty (Cox theorem, 1946, and Caticha, 2012, pp.7-26).

- Sum rule. The disjunction for two events A and B given background I is given by

$$p(A+B|I) = p(A|I) + p(B|I) - p(A,B|I)$$

- Product rule. The conjuction of two events A and B with given background I is given by

$$p(A,B|I) = p(A|B,I)p(B|I)$$

- All legitimate probabilistic relations can be derived from the sum and product rules!

Independent and Mutually Exclusive Events

- Two events A and B are said to be independent if the probability of one is not altered by information about the truth of the other, i.e., $p(A|B) = p(A)$
  - ⇒ If A and B are independent, given I, then the product rule simplifies to

$$p(A,B|I) = p(A|I)p(B|I)$$

- Two events A_1 and A_2 are said to be mutually exclusive if they cannot be true simultanously, i.e., if $p(A_1,A_2) = 0$.
  - ⇒ For mutually exclusive events, the sum rule simplifies to

$$p(A_1+A_2) = p(A_1) + p(A_2)$$

- A set of events $A_1, A_2, \ldots, A_N$ is said to be collectively exhaustive if one of the statements is necessarily true, i.e., $A_1 + A_2 + \cdots + A_N = \text{TRUE}$, or equivalently

$$p(A_1 + A_2 + \cdots + A_N) = 1$$

- Note that, if $A_1, A_2, \ldots, A_n$ are both mutually exclusive and collectively exhausitive (MECE) events, then

$$\sum_{n=1}^{N} p(A_n) = p(A_1 + \ldots + A_N) = 1$$

  - More generally, if $\{A_n\}$ are MECE events, then $\sum_{n=1}^{N} p(A_n, B) = p(B)$

The Sum Rule and Marginalization

- We mentioned that every inference problem in PT can be evaluated through the sum and product rules. Next, we present two useful corollaries: (1) Marginalization and (2) Bayes rule

- If $X \in$ and $Y \in$ are random variables over finite domains, than it follows from the above considerations about MECE events that

$$\sum_{Y \in} p(X, Y) = p(X).$$

- Summing Y out of a joint distribution $p(X, Y)$ is called marginalization and the result $p(X)$ is sometimes referred to as the marginal probability.

- Note that this is just a generalized sum rule. In fact, Bishop (p.14) (and some other authors as well) calls this the sum rule.

- Of course, in the continuous domain, the (generalized) sum rule becomes

$$p(X) = \int p(X, Y)\, \mathrm{d}Y$$

The Product Rule and Bayes Rule

- Consider two variables D and θ; it follows from symmetry arguments that

$$p(D, \theta) = p(D \mid \theta) p(\theta) = p(\theta \mid D) p(D)$$

  and hence that

$$p(\theta \mid D) = \frac{p(D \mid \theta)}{p(D)} p(\theta).$$

- This formula is called Bayes rule (or Bayes theorem). While Bayes rule is always true, a particularly useful application occurs when D refers to an observed data set and θ is set of model parameters. In that case,
  - the prior probability $p(\theta)$ represents our state-of-knowledge about proper values for θ, before seeing the data D.
  - the posterior probability $p(\theta|D)$ represents our state-of-knowledge about θ after we have seen the data.

⇒ Bayes rule tells us how to update our knowledge about model parameters when facing new data. Hence,

Bayes rule is the fundamental rule for learning from data!

Bayes Rule Nomenclature

- Some nomenclature associated with Bayes rule:

$$\underset{\text{posterior}}{p(\theta|D)} = \frac{\overset{\text{likelihood}}{p(D|\theta)} \times \overset{\text{prior}}{p(\theta)}}{\underset{\text{evidence}}{p(D)}}$$

- Note that the evidence (a.k.a. marginal likelihood) can be computed from the numerator through marginalization since

$$p(D) = \int p(D,\theta)\,\mathrm{d}\theta = \int p(D|\theta)p(\theta)\,\mathrm{d}\theta$$

- Hence, having access to likelihood and prior is in principle sufficient to compute both the evidence and the posterior. To emphasize that point, Bayes rule is sometimes written as a transformation:

$$\underset{\text{posterior}}{p(\theta|D)} \cdot \underset{\text{evidence}}{p(D)} = \underset{\text{likelihood}}{p(D|\theta)} \cdot \underset{\text{prior}}{p(\theta)}$$

this is what we want to compute — this is available

- For a given data set D, the posterior probabilities of the parameters scale relatively against each other as

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

- ⇒ All that we can learn from the observed data is contained in the likelihood function $p(D|\theta)$. This is called the likelihood principle.

The Likelihood Function vs the Sampling Distribution

- Consider a distribution $p(D|\theta)$, where D relates to variables that are observed (i.e., a "data set") and θ are model parameters.

- In general, $p(D|\theta)$ is just a function of the two variables D and θ. We distinguish two interpretations of this function, depending on which variable is observed (or given by other means).

- The sampling distribution (a.k.a. the data-generating distribution)

  $$p(D|\theta = \theta_0)$$

  (which is a function of D only) describes a probability distribution for data D, assuming that it is generated by the given model with parameters fixed at $\theta = \theta_0$.

- In a machine learning context, often the data is observed, and θ is the free variable. In that case, for given observations $D = D_0$, the likelihood function (which is a function only of the model parameters θ) is defined as

  $$\mathrm{L}(\theta) \triangleq p(D = D_0|\theta)$$

- Note that $\mathrm{L}(\theta)$ is not a probability distribution for θ since in general $\sum_\theta \mathrm{L}(\theta) \neq 1$.

Code Example: Sampling Distribution and Likelihood Function for the Coin Toss

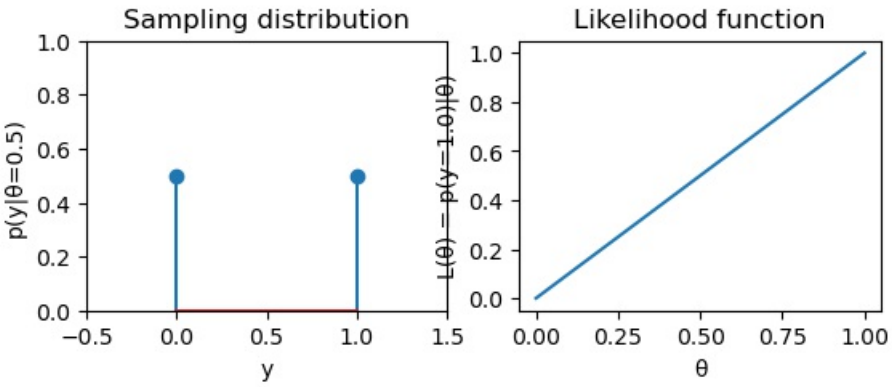
Consider the following simple model for the outcome (head or tail) $y \in \{0, 1\}$ of a biased coin toss with parameter $\theta \in [0, 1]$:

$$p(y|\theta) \triangleq \theta^y(1-\theta)^{1-y}$$

We can plot both the sampling distribution $p(y|\theta = 0.8)$ and the likelihood function $L(\theta) = p(y = 0|\theta)$.

```
using Pkg; Pkg.activate("probprog/workspace");Pkg.instantiate();
IJulia.clear_output();
```

```
using PyPlot
#using Plots
p(y,θ) = θ.^y .* (1 .- θ).^(1 .- y)
f = figure()

θ = 0.5 # Set parameter
# Plot the sampling distribution
subplot(221); stem([0,1], p([0,1],θ));
title("Sampling distribution");
xlim([-0.5,1.5]); ylim([0,1]); xlabel("y"); ylabel("p(y|θ=$(θ))");

subplot(222);
_θ = 0:0.01:1
y = 1.0 # Plot p(y=1 | θ)
plot(_θ,p(y,_θ))
title("Likelihood function");
xlabel("θ");
ylabel("L(θ) = p(y=$y)|θ)");
```



The (discrete) sampling distribution is a valid probability distribution. However, the likelihood function $L(\theta)$ clearly isn't, since $\int_0^1 L(\theta)\mathrm{d}\theta \neq 1$.

Probabilistic Inference

- Probabilistic inference refers to computing

$$p(\text{ whatever-we-want-to-know } | \text{ whatever-we-already-know })$$

  - For example:

$$p(\text{ Mr.S.-killed-Mrs.S. } | \text{ he-has-her-blood-on-his-shirt})$$
$$p(\text{ transmitted-codeword } | \text{ received-codeword})$$

- This can be accomplished by repeated application of sum and product rules.

- In particular, consider a joint distribution $p(X, Y, Z)$. Assume we are interested in $p(X|Z)$:

$$p(X|Z) \overset{p}{=} \frac{p(X,Z)}{p(Z)} \overset{s}{=} \frac{\sum_Y p(X,Y,Z)}{\sum_{X,Y} p(X,Y,Z)},$$

  where the 's' and 'p' above the equality sign indicate whether the sum or product rule was used.

- In the rest of this course, we'll encounter many long probabilistic derivations. For each manipulation, you should be able to associate an 's' (for sum rule), a 'p' (for product or Bayes rule) or an 'm' (for a simplifying model assumption) above any equality sign.

Working out the example problem: Disease Diagnosis

- Problem: Given a disease D with prevalence of 1% and a test procedure T with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85%, what is the chance that somebody who tests positive actually has the disease?

- Solution: The given data are $p(D=1) = 0.01$, $p(T=1|D=1) = 0.95$ and $p(T=0|D=0) = 0.85$. Then according to Bayes rule,

$$
\begin{aligned}
& p(D=1|T=1) \\
& \overset{p}{=} \frac{p(T=1|D=1)p(D=1)}{p(T=1)} \\
& \overset{s}{=} \frac{p(T=1|D=1)p(D=1)}{p(T=1|D=1)p(D=1)+p(T=1|D=0)p(D=0)} \\
& = \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.15 \times 0.99} = 0.0601
\end{aligned}
$$

Inference Exercise: Bag Counter

- Problem: A bag contains one ball, known to be either white or black. A white ball is put in, the bag is shaken, and a ball is drawn out, which proves to be white. What is now the chance of drawing a white ball?

- Solution: Again, use Bayes and marginalization to arrive at $p(\text{white}|\text{data}) = 2/3$, see the Exercises notebook.
- ⇒ Note that probabilities describe a person's state of knowledge rather than a 'property of nature'.

Inference Exercise: Causality?

- Problem: A dark bag contains five red balls and seven green ones. (a) What is the probability of drawing a red ball on the first draw? Balls are not returned to the bag after each draw. (b) If you know that on the second draw the ball was a green one, what is now the probability of drawing a red ball on the first draw?

- Solution: (a) $5/12$. (b) $5/11$, see the Exercises notebook.
- ⇒ Again, we conclude that conditional probabilities reflect implications for a state of knowledge rather than temporal causality.

Linear Transformations

- Consider an arbitrary distribution $p(X)$ with mean μ_x and variance Σ_x and the linear transformation

$$Z = AX + b\,.$$

- No matter the specification of $p(X)$, we can derive that (see Exercises notebook)

$$\mu_z = A\mu_x + b$$
$$\Sigma_z = A\,\Sigma_x A^T$$

  - (The tag (SRG-3a) refers to the corresponding eqn number in Sam Roweis Gaussian identities notes.)

PDF for the Sum of Two Variables

- Given eqs SRG-3a and SRG-3b (previous cell), you should now be able to derive the following: for any distribution of variable X and Y and sum $Z = X + Y$ (proof by Exercise)

$$\mu_z = \mu_x + \mu_y$$
$$\Sigma_z = \Sigma_x + \Sigma_y + 2\Sigma_{xy}$$

- Clearly, it follows that if X and Y are independent, then

$$\Sigma_z = \Sigma_x + \Sigma_y$$

- More generally, given two independent variables X and Y, with PDF's $p_x(x)$ and $p_y(y)$. The PDF $p_z(z)$for $Z = X + Y$ is given by the convolution

$$p_z(z) = \int_{-\infty}^{\infty} p_x(x)p_y(z-x)\,\mathrm{d}x$$

- Proof: Let $p_z(z)$ be the probability that Z has value z. This occurs if X has some value x and at the same time $Y = z - x$, with joint probability $p_x(x)p_y(z-x)$. Since x can be any value, we sum over all possible values for x to get $p_z(z) = \int_{-\infty}^{\infty} p_x(x)p_y(z-x)\,\mathrm{d}x$
  - Note that $p_z(z) \neq p_x(x) + p_y(y)$!!

- https://en.wikipedia.org/wiki/List_of_convolutions_of_probability_distributions shows how these convolutions work out for a few common probability distributions.

- In linear stochastic systems theory, the Fourier Transform of a PDF (i.e., the characteristic function) plays an important computational role. Why?

Code Example: Sum of Two Gaussian Distributed Variables

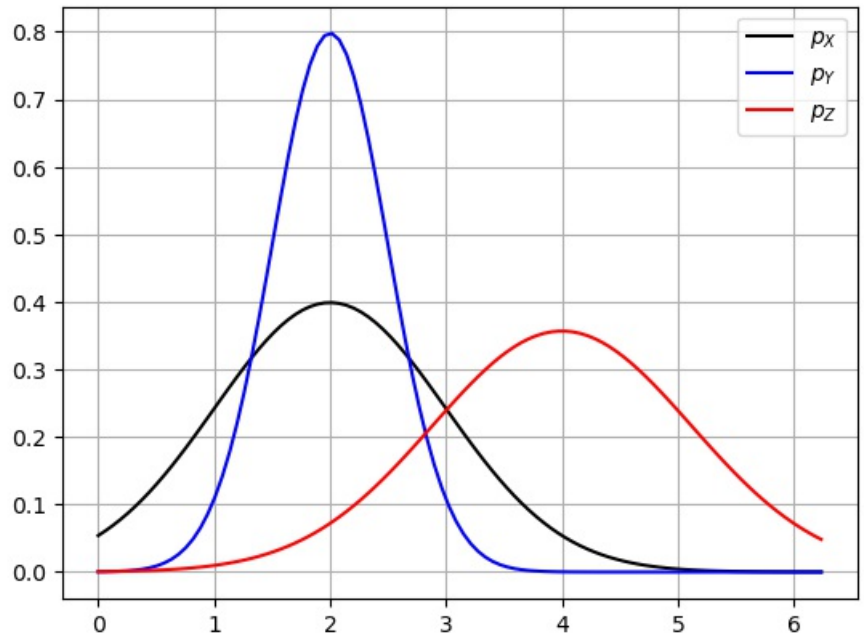
- Consider the PDF of the sum of two independent Gaussian distributed X and Y:

$$p_X(x) = \quad (x \mid \mu_X, \sigma_X^2)$$
$$p_Y(y) = \quad (y \mid \mu_Y, \sigma_Y^2)$$

- Let $Z = X + Y$. Performing the convolution (nice exercise) yields a Gaussian PDF for Z:

$$p_Z(z) = \quad (z \mid \mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2).$$

```
using PyPlot, Distributions
μx = 2.
σx = 1.
μy = 2.
σy = 0.5
μz = μx+μy; σz = sqrt(σx^2 + σy^2)
x = Normal(μx, σx)
y = Normal(μy, σy)
z = Normal(μz, σz)
range_min = minimum([μx-2*σx, μy-2*σy, μz-2*σz])
range_max = maximum([μx+2*σx, μy+2*σy, μz+2*σz])
range_grid = range(range_min, stop=range_max, length=100)
plot(range_grid, pdf.(x,range_grid), "k-")
plot(range_grid, pdf.(y,range_grid), "b-")
plot(range_grid, pdf.(z,range_grid), "r-")
legend([L"p_X", L"p_Y", L"p_Z"])
grid()
```



PDF for the Product of Two Variables

- For two continuous random independent variables X and Y, with PDF's $p_x(x)$ and $p_y(y)$, the PDF of $Z = XY$ is given by

$$p_z(z) = \int_{-\infty}^{\infty} p_x(x)\, p_y(z/x)\, \frac{1}{|x|}\, \mathrm{d}x$$

- For proof, see https://en.wikipedia.org/wiki/Product_distribution

- Generally, this integral does not lead to an analytical expression for $p_z(z)$. For example, the product of two independent variables that are both normally (Gaussian) distributed does not lead to a normal distribution.
  - Exception: the distribution of the product of two variables that both have log-normal distributions is again a lognormal distribution.
    - (If X has a normal distribution, then $Y = \exp(X)$ has a log-normal distribution.)

Moments of the PDF

- Consider a distribution $p(x)$. The expected value or mean is defined as

$$\mu_x = \mathrm{E}[x] \triangleq \int x\, p(x)\, \mathrm{d}x$$

- The variance of x is defined as

$$\Sigma_x \triangleq \mathrm{E}\left[(x - \mu_x)(x - \mu_x)^T\right]$$

- The covariance matrix between vectors x and y is defined as

$$\begin{aligned}\Sigma_{xy} &\triangleq \mathrm{E}\left[(x - \mu_x)(y - \mu_y)^T\right] \\ &= \mathrm{E}\left[(x - \mu_x)(y^T - \mu_y^T)\right] \\ &= \mathrm{E}[xy^T] - \mu_x \mu_y^T\end{aligned}$$

  - Clearly, if x and y are independent, then $\Sigma_{xy} = 0$, since $\mathrm{E}[xy^T] = \mathrm{E}[x]\mathrm{E}[y^T] = \mu_x \mu_y^T$.

Variable Transformations

- Suppose x is a discrete random variable with probability mass function $P_x(x)$, and $y = h(x)$ is a one-to-one function with $x = g(y) = h^{-1}(y)$. Then

$$P_y(y) = P_x(g(y)).$$

- Proof: $P_y(\hat{y}) = P(y = \hat{y}) = P(h(x) = \hat{y}) = P(x = g(\hat{y})) = P_x(g(\hat{y}))$. □

- If x is defined on a continuous domain, and $p_x(x)$ is a probability density function, then probability mass is represented by the area under a (density) curve. Let $a = g(c)$ and $b = g(d)$. Then

$$\begin{aligned} P(a \le x \le b) &= \int_a^b p_x(x)\mathrm{d}x \\ &= \int_{g(c)}^{g(d)} p_x(x)\mathrm{d}x \\ &= \int_c^d p_x(g(y))\mathrm{d}g(y) \\ &= \int_c^d p_x(g(y))g'(y)\mathrm{d}y \\ &= \underbrace{P(c \le y \le d)}_{p_y(y)} \end{aligned}$$

- Equating the two probability masses leads to identificaiton of the relation

$$p_y(y) = p_x(g(y))g'(y),$$

  which is also known as the Change-of-Variable theorem.

- If the tranformation $y = h(x)$ is not invertible, then $x = g(y)$ does not exist. In that case, you can still work out the transformation by equating equivalent probability masses in the two domains.

Example: Transformation of a Gaussian Variable

- Let $p_x(x) = \quad (x \mid \mu, \sigma^2)$ and $y = \frac{x-\mu}{\sigma}$.
- Problem: What is $p_y(y)$?
- Solution: Note that $h(x)$ is invertible with $x = g(y) = \sigma y + \mu$. The change-of-variable formula leads to

$$\begin{aligned} p_y(y) &= p_x(g(y)) \cdot g'(y) \\ &= p_x(\sigma y + \mu) \cdot \sigma \\ &= \frac{1}{\sigma\sqrt{(2\pi)}}\exp\left(-\frac{(\sigma y + \mu - \mu)^2}{2\sigma^2}\right) \cdot \sigma \\ &= \frac{1}{\sqrt{(2\pi)}}\exp\left(-\frac{y^2}{2}\right) \\ &= \quad (y \mid 0, 1) \end{aligned}$$

Summary

- Probabilities should be interpretated as degrees of belief, i.e., a state-of-knowledge, rather than a property of nature.

- We can do everything with only the sum rule and the product rule. In practice, Bayes rule and marginalization are often very useful for inference, i.e., for computing

$$p(\text{ what-we-want-to-know } \mid \text{ what-we-already-know }).$$

- Bayes rule

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

  is the fundamental rule for learning from data!

- For a variable X with distribution $p(X)$ with mean μ_x and variance Σ_x, the mean and variance of the Linear Transformation $Z = AX + b$ is given by

$$\mu_z = A\mu_x + b$$
$$\Sigma_z = A\,\Sigma_x A^T$$

- That's really about all you need to know about probability theory, but you need to really know it, so do the Exercises.

Bayesian Machine Learning

Preliminaries

- Goals
  - Introduction to Bayesian (i.e., probabilistic) modeling
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 68-74 (on the coin toss example)
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.35-44 (section 2.9, on deriving Bayes rule for updating probabilities)
    - David Blei - 2014 - Build, Compute, Critique, Repeat: Data Analysis with Latent Variable Models, on the Build-Compute-Critique-Repeat design model.

Challenge: Predicting a Coin Toss

- Problem: We observe a the following sequence of heads (h) and tails (t) when tossing the same coin repeatedly

$$D = \{hthhtth\}\,.$$

- What is the probability that heads comes up next?

- Solution: later in this lecture.

The Bayesian Machine Learning Framework

- Suppose that your application is to predict a future observation x, based on N past observations $D = \{x_1, \ldots, x_N\}$.

- The Bayesian design approach to solving this task involves four stages:

REPEAT

> 1- Model specification
> 2- Parameter estimation (i.e., learning from an observed data set using Bayesian inference)
> 3- Model evaluation (how good is this (trained) model?)

UNTIL model performance is satisfactory

> 4- Apply model, e.g. for prediction or classification of new data

- In principle, based on the model evaluation results, you may want to re-specify your model and repeat the design process (a few times), until model performance is acceptable.

- Next, we discuss these four stages in a bit more detail.

(1) Model specification

- Your first task is to propose a probabilistic model (m) for generating the observations x.

- A probabilistic model m consists of a joint distribution $p(x, \theta \,|\, m)$ that relates observations x to model parameters θ. Usually, the model is proposed in the form of a data generating distribution $p(x \,|\, \theta, m)$ and a prior $p(\theta \,|\, m)$.

- You are responsible to choose the data generating distribution $p(x \,|\, \theta)$ based on your physical understanding of the data generating process. (For simplicity, we dropped the given dependency on m from the notation).

- You must also choose the prior $p(\theta)$ to reflect what you know about the parameter values before you see the data D.

(2) Parameter estimation

- Note that, for a given data set $D = \{x_1, x_2, \ldots, x_N\}$ with independent observations x_n, the likelihood is

$$p(D \,|\, \theta) = \prod_{n=1}^{N} p(x_n \,|\, \theta),$$

  so usually you select a model for generating one observation x_n and then use (in-)dependence assumptions to combine these models into a likelihood function for the model parameters.

- The likelihood and prior both contain information about the model parameters. Next, you use Bayes rule to fuse these two information sources into a posterior distribution for the parameters,

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \propto p(D|\theta)p(\theta)$$

- Note that there's no need for you to design some clever parameter estimation algorithm. Bayes rule is the parameter estimation algorithm. The only complexity lies in the computational issues!

- This "recipe" works only if the right-hand side (RHS) factors can be evaluated; the computational details can be quite challenging and this is what machine learning is about.
- ⇒ Machine learning is EASY, apart from computational details :)

(3) Model Evaluation

- In the framework above, parameter estimation was executed by "perfect" Bayesian reasoning. So is everything settled now?

- No, there appears to be one remaining problem: how good really were our model assumptions $p(x|\theta)$ and $p(\theta)$? We want to "score" the model performance.

- Note that this question is only interesting if we have alternative models to choose from.

- Let's assume that we have more candidate models, say $= \{m_1, \ldots, m_K\}$ where each model relates to specific prior $p(\theta|m_k)$ and likelihood $p(D|\theta, m_k)$? Can we evaluate the relative performance of a model against another model from the set?

- Start again with model specification. You must now specify a prior $p(m_k)$ (next to the likelihood $p(D|\theta, m_k)$ and prior $p(\theta|m_k)$) for each of the models and then solve the desired inference problem:

$$p(m_k|D) = \frac{p(D|m_k)p(m_k)}{p(D)}$$

$$\begin{aligned} \text{model posterior} &\propto p(m_k) \cdot p(D|m_k) \\ &= p(m_k) \cdot \int_\theta p(D, \theta|m_k)\,\mathrm{d}\theta \\ &= p(m_k) \cdot \int_\theta p(D|\theta, m_k)\,p(\theta|m_k)\,\mathrm{d}\theta \end{aligned}$$

model prior · likelihood · prior

evidence $p(D|m_k)$ = model likelihood

- This procedure is called Bayesian model comparison, which requires that you calculate the "evidence" (= model likelihood).

- ⇒ In a Bayesian framework, model estimation follows the same recipe as parameter estimation; it just works at one higher hierarchical level. Compare the required calulations:

$$p(\theta|D) \propto p(D|\theta)p(\theta) \qquad \text{(parameter estimation)}$$
$$p(m_k|D) \propto p(D|m_k)p(m_k) \qquad \text{(model comparison)}$$

- In principle, you could proceed with asking how good your choice for the candidate model set was. You would have to provide a set of alternative model sets $\{ _1, _2, \ldots, _M\}$ with priors $p(_m)$ for each set and compute posteriors $p(_m|D)$. And so forth ...

- With the (relative) performance evaluation scores of your model in hand, you could now re-specify your model (hopefully an improved model) and repeat the design process until the model performance score is acceptable.

- As an aside, in the (statistics and machine learning) literature, performance comparison between two models is often reported by the Bayes Factor, which is defined as the ratio of model evidences:

$$\underbrace{\frac{p(D|m_1)}{p(D|m_2)}}_{\text{Bayes Factor}} = \frac{\frac{p(D,m_1)}{p(m_1)}}{\frac{p(D,m_2)}{p(m_2)}}$$
$$= \frac{p(D,m_1)}{p(m_1)} \cdot \frac{p(m_2)}{p(D,m_2)}$$
$$= \frac{p(m_1|D)p(D)}{p(m_1)} \cdot \frac{p(m_2)}{p(m_2|D)p(D)}$$
$$= \underbrace{\frac{p(m_1|D)}{p(m_2|D)}}_{\substack{\text{posterior} \\ \text{ratio}}} \cdot \underbrace{\frac{p(m_2)}{p(m_1)}}_{\substack{\text{prior} \\ \text{ratio}}}$$

  - Hence, for equal model priors ($p(m_1) = p(m_2) = 0.5$), the Bayes Factor reports the posterior probability ratio for the two models.

(4) Prediction

- Once we are satisfied with the evidence for a (trained) model, we can apply the model to our prediction/classification/etc task.

- Given the data D, our knowledge about the yet unobserved datum x is captured by (everything is conditioned on the selected model)

$$p(x|D) \overset{s}{=} \int p(x, \theta|D)\, \mathrm{d}\theta$$
$$\overset{p}{=} \int p(x|\theta, D)p(\theta|D)\, \mathrm{d}\theta$$
$$\overset{m}{=} \int \underbrace{p(x|\theta)}_{\text{data generation dist.}} \cdot \underbrace{p(\theta|D)}_{\text{posterior}}\, \mathrm{d}\theta$$

- In the last equation, the simplification $p(x \mid \theta, D) = p(x \mid \theta)$ follows from our model specification. We assumed a parametric data generating distribution $p(x \mid \theta)$ with no explicit dependency on the data set D.

- Again, no need to invent a special prediction algorithm. Probability theory takes care of all that. The complexity of prediction is just computational: how to carry out the marginalization over θ.

- Note that the application of the learned posterior $p(\theta \mid D)$ not necessarily has to be a prediction task. We use it here as an example, but other applications (e.g., classification, regression etc.) are of course also possible.

- What did we learn from D? Without access to D, we would predict new observations through

$$p(x) = \int p(x, \theta)\, \mathrm{d}\theta = \int p(x \mid \theta) \cdot \underbrace{p(\theta)}_{\text{prior}}\, \mathrm{d}\theta$$

Prediction with multiple models

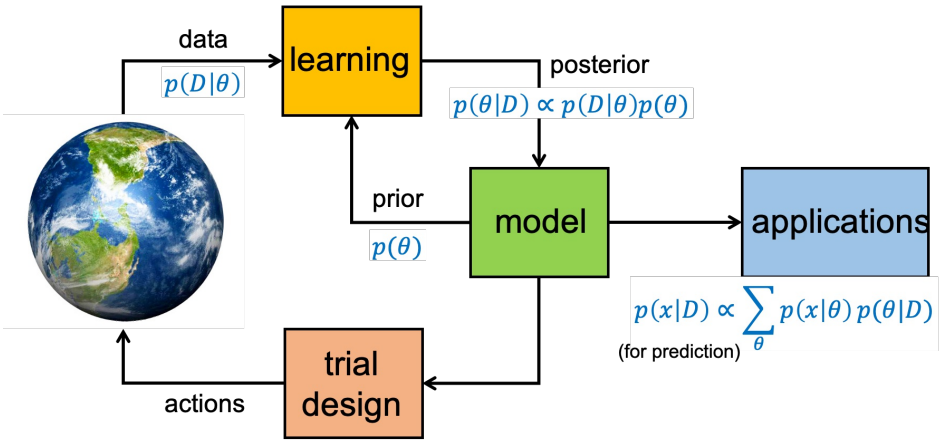
- When you have a posterior $p(m_k \mid D)$ for the models, you don't need to choose one model for the prediction task. You can do prediction by Bayesian model averaging to utilitize the predictive power from all models:

$$\begin{aligned} p(x \mid D) &= \sum_k \int p(x, \theta, m_k \mid D)\, \mathrm{d}\theta \\ &= \sum_k \int p(x \mid \theta, m_k)\, p(\theta \mid m_k, D)\, p(m_k \mid D)\, \mathrm{d}\theta \\ &= \sum_k \underbrace{p(m_k \mid D)}_{\substack{\text{model} \\ \text{posterior}}} \cdot \int \underbrace{p(\theta \mid m_k, D)}_{\substack{\text{parameter} \\ \text{posterior}}} \underbrace{p(x \mid \theta, m_k)}_{\substack{\text{data generating} \\ \text{distribution}}} \mathrm{d}\theta \end{aligned}$$

- Alternatively, if you do need to work with one model (e.g. due to computational resource constraints), you can for instance select the model with largest posterior $p(m_k \mid D)$ and use that model for prediction. This is called Bayesian model selection.

Bayesian Machine Learning and the Scientific Method Revisited

- The Bayesian design process provides a unified framework for the Scientific Inquiry method. We can now add equations to the design loop. (Trial design to be discussed in Intelligent Agent lesson.)



Now Solve the Example Problem: Predicting a Coin Toss

- We observe a the following sequence of heads (h) and tails (t) when tossing the same coin repeatedly

$$D = \{hthhtth\}\,.$$

- What is the probability that heads comes up next? We solve this in the next slides ...

Coin toss example (1): Model Specification

- We observe a sequence of N coin tosses $D = \{x_1, \ldots, x_N\}$ with n heads.

- Let us denote outcomes by

$$x_k = \begin{cases} h & \text{if heads comes up} \\ t & \text{if tails} \end{cases}$$

Likelihood

- Assume a Bernoulli distributed variable $p(x_k = h\,|\,\mu) = \mu$, which leads to a binomial distribution for the likelihood (assume n times heads were thrown):

$$p(D\,|\,\mu) = \prod_{k=1}^{N} p(x_k\,|\,\mu) = \mu^n (1-\mu)^{N-n}$$

Prior

- Assume the prior belief is governed by a beta distribution

$$p(\mu) = \text{Beta}(\mu \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1}$$

where the Gamma function is sort of a generalized factorial function. If α, β are integers, then

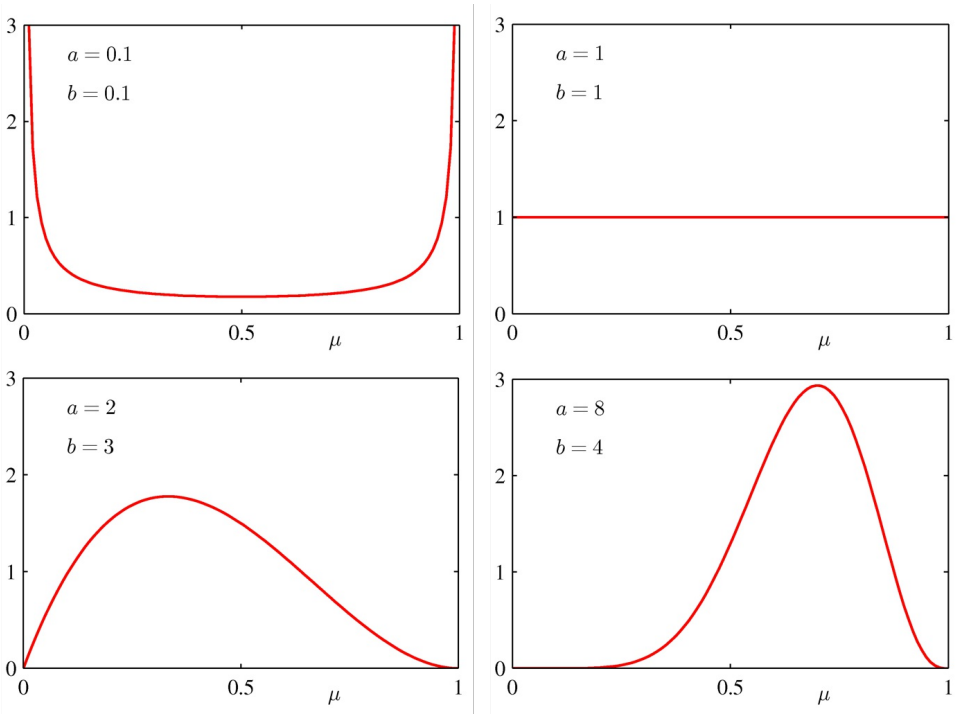
$$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)(\Gamma(\beta)} = \frac{(\alpha+\beta-1)!}{(\alpha-1)!(\beta-1)!}$$

- A what distribution? Yes, the beta distribution is a conjugate prior for the binomial distribution, which means that

$$\underset{\text{posterior}}{\text{beta}} \propto \underset{\text{likelihood}}{\text{binomial}} \times \underset{\text{prior}}{\text{beta}}$$

  so we get a closed-form posterior.

- α and β are called hyperparameters, since they parameterize the distribution for another parameter (μ). E.g., $\alpha = \beta = 1$ (uniform).



- (Bishop Fig.2.2). Plots of the beta distribution $\text{Beta}(\mu \mid a, b)$ as a function of μ for various values of the hyperparameters a and b.

Coin toss example (2): Parameter estimation

- Infer posterior PDF over μ through Bayes rule

$$\begin{aligned} p(\mu \mid D) &\propto p(D \mid \mu) p(\mu \mid \alpha, \beta) \\ &= \mu^n (1-\mu)^{N-n} \times \mu^{\alpha-1} (1-\mu)^{\beta-1} \\ &= \mu^{n+\alpha-1} (1-\mu)^{N-n+\beta-1} \end{aligned}$$

hence the posterior is also beta-distributed as

$$p(\mu \mid D) = \text{Beta}(\mu \mid n+\alpha, N-n+\beta)$$

Coin Toss Example (3): Prediction

- For simplicity, we skip the model evaluation task here and proceed to apply the trained model. Let's use it to predict future observations.

- Marginalize over the parameter posterior to get the predictive PDF for a new coin toss $x_.$, given the data D,

$$\begin{aligned} p(x_. = h \mid D) &= \int_0^1 p(x_. = h \mid \mu)\, p(\mu \mid D)\, \mathrm{d}\mu \\ &= \int_0^1 \mu \times \text{Beta}(\mu \mid n+\alpha, N-n+\beta)\, \mathrm{d}\mu \\ &= \frac{n+\alpha}{N+\alpha+\beta} \end{aligned}$$

- This result is known as Laplace's rule of succession

- The above integral computes the mean of a beta distribution, which is given by $\mathrm{E}[x] = \frac{a}{a+b}$ for $x \sim \text{Beta}(a, b)$, see wikipedia.

- Finally, we're ready to solve our example problem: for $D = \{hthhtth\}$ and uniform prior ($\alpha = \beta = 1$), we get

$$p(x_. = h \mid D) = \frac{n+1}{N+2} = \frac{4+1}{7+2} = \frac{5}{9}$$

Coin Toss Example: What did we learn?

- What did we learn from the data? Before seeing any data, we think that

$$p(x_. = h) = p(x_. = h \mid D)\Big|_{n=N=0} = \frac{\alpha}{\alpha+\beta}.$$

- Hence, α and β are prior pseudo-counts for heads and tails respectively.

- After the N coin tosses, we think that $p(x_{\cdot} = h \mid D) = \frac{n+\alpha}{N+\alpha+\beta}$.

- Note the following decomposition

$$
\begin{aligned}
p(x_{\cdot} = h \mid D) &= \frac{n+\alpha}{N+\alpha+\beta} = \frac{n}{N+\alpha+\beta} + \frac{\alpha}{N+\alpha+\beta} \\
&= \frac{N}{N+\alpha+\beta} \cdot \frac{n}{N} + \frac{\alpha+\beta}{N+\alpha+\beta} \cdot \frac{\alpha}{\alpha+\beta} \\
&= \underbrace{\frac{\alpha}{\alpha+\beta}}_{\substack{\text{prior} \\ \text{prediction}}} + \underbrace{\frac{N}{N+\alpha+\beta}}_{\text{gain}} \cdot \underbrace{\left(\underbrace{\frac{n}{N}}_{\substack{\text{data-based} \\ \text{prediction}}} - \underbrace{\frac{\alpha}{\alpha+\beta}}_{\substack{\text{prior} \\ \text{prediction}}}\right)}_{\text{prediction error}}
\end{aligned}
$$

correction

- Note that, since $0 \leq \underbrace{\frac{N}{N+\alpha+\beta}}_{\text{gain}} < 1$, the Bayesian prediction lies between (fuses) the prior and data-based predictions. The data plays the role of "correcting" the prior prediction.

- For large N, the gain goes to 1 and $p(x_{\cdot} = h \mid D)\Big|_{N\to\infty} \to \frac{n}{N}$ goes to the data-based prediction (the observed relative frequency).

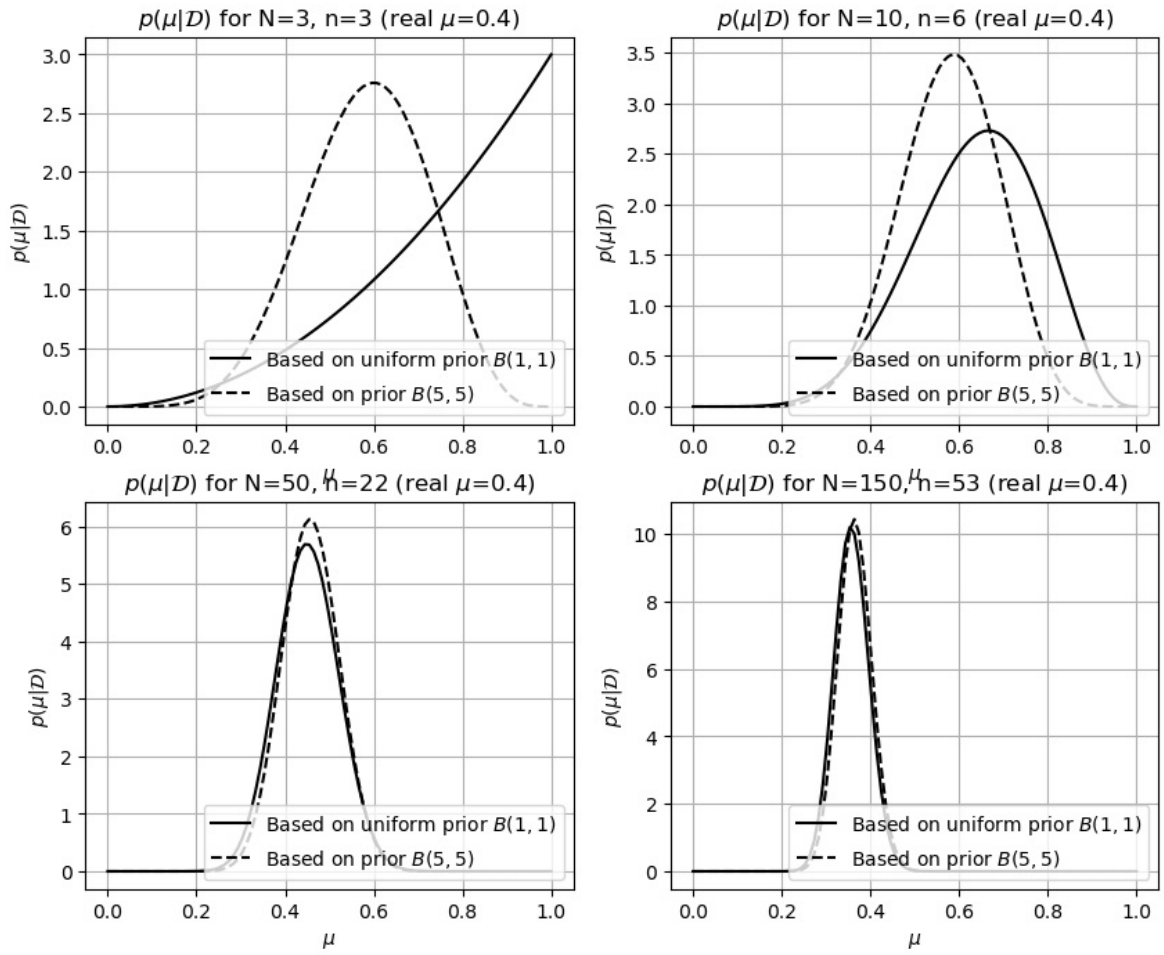
CODE EXAMPLE

Bayesian evolution of $p(\mu \mid D)$ for the coin toss

Let's see how $p(\mu \mid D)$ evolves as we increase the number of coin tosses N. We'll use two different priors to demonstrate the effect of the prior on the posterior (set $N = 0$ to inspect the prior).

```
using Pkg; Pkg.activate("probprog/workspace");Pkg.instantiate();
IJulia.clear_output();
```

```
using PyPlot, Distributions
f = figure()
range_grid = range(0.0, stop=1.0, length=100)
μ = 0.4
samples = rand(192) .<= μ # Flip 192 coins
posterior1 = Array{Distribution}(undef,193)
posterior2 = Array{Distribution}(undef,193)
for N=0:1:192
    n = sum(samples[1:N]) # Count number of heads in first N flips
    posterior1[N+1] = Beta(1+n, 1+(N-n))
    posterior2[N+1] = Beta(5+n, 5+(N-n))

end

fig = figure("Posterior distributions", figsize=(10,8));
ax1 = fig.add_subplot(2,2,1);
ax2 = fig.add_subplot(2,2,2);
ax3 = fig.add_subplot(2,2,3);
ax4 = fig.add_subplot(2,2,4);
plt.subplot(ax1); plot(range_grid,pdf.(posterior1[3],range_grid), "k-");
plt.subplot(ax1); plot(range_grid,pdf.(posterior2[3],range_grid), "k--");
xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
title(L"p(\mu|\mathcal{D})"*" for N=$(3), n=$(sum(samples[1:3])) (real \$\\mu\$=$(μ))")
legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4)

plt.subplot(ax2); plot(range_grid,pdf.(posterior1[10],range_grid), "k-");
plt.subplot(ax2); plot(range_grid,pdf.(posterior2[10],range_grid), "k--");
xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
title(L"p(\mu|\mathcal{D})"*" for N=$(10), n=$(sum(samples[1:10])) (real \$\\mu\$=$(μ))")
legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4)

plt.subplot(ax3); plot(range_grid,pdf.(posterior1[50],range_grid), "k-");
plt.subplot(ax3); plot(range_grid,pdf.(posterior2[50],range_grid), "k--");
xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
title(L"p(\mu|\mathcal{D})"*" for N=$(50), n=$(sum(samples[1:50])) (real \$\\mu\$=$(μ))")
legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4)


plt.subplot(ax4); plot(range_grid,pdf.(posterior1[150],range_grid), "k-");
plt.subplot(ax4); plot(range_grid,pdf.(posterior2[150],range_grid), "k--");
xlabel(L"\mu"); ylabel(L"p(\mu|\mathcal{D})"); grid()
title(L"p(\mu|\mathcal{D})"*" for N=$(150), n=$(sum(samples[1:150])) (real \$\\mu\$=$(μ))")
legend(["Based on uniform prior "*L"B(1,1)","Based on prior "*L"B(5,5)"], loc=4);
```



⇒ With more data, the relevance of the prior diminishes!

From Posterior to Point-Estimate

- In the example above, Bayesian parameter estimation and prediction were tractable in closed-form. This is often not the case. We will need to approximate some of the computations.

- Recall Bayesian prediction

$$p(x \mid D) = \int p(x \mid \theta) p(\theta \mid D) \, \mathrm{d}\theta$$

- If we approximate posterior $p(\theta \mid D)$ by a delta function for one 'best' value $\hat{\theta}$, then the predictive distribution collapses to

$$p(x \mid D) = \int p(x \mid \theta) \, \delta(\theta - \hat{\theta}) \, \mathrm{d}\theta = p(x \mid \hat{\theta})$$

- This is just the data generating distribution $p(x \mid \theta)$ evaluated at $\theta = \hat{\theta}$, which is easy to evaluate.

- The next question is how to get the parameter estimate $\hat{\theta}$? (See next slide).

Some Well-known Point-Estimates

- Bayes estimate (the mean of the posterior)

$$\hat{\theta}_{bayes} = \int \theta p(\theta \mid D)\, \mathrm{d}\theta$$

- Maximum A Posteriori (MAP) estimate

$$\hat{\theta}_{\text{map}} = \arg\max_{\theta} p(\theta \mid D) = \arg\max_{\theta} p(D \mid \theta) p(\theta)$$

- Maximum Likelihood (ML) estimate

$$\hat{\theta}_{ml} = \arg\max_{\theta} p(D \mid \theta)$$

  - Note that Maximum Likelihood is MAP with uniform prior
  - ML is the most common approximation to the full Bayesian posterior.

Bayesian vs Maximum Likelihood Learning

Consider the task: predict a datum x from an observed data set D.

	Bayesian	Maximum Likelihood
1. Model Specification	Choose a model m with data generating distribution $p(x \mid \theta, m)$ and parameter prior $p(\theta \mid m)$	Choose a model m with same data generating distribution $p(x \mid \theta, m)$. No need for priors.
2. Learning	use Bayes rule to find the parameter posterior, $p(\theta \mid D) \propto p(D \mid \theta) p(\theta)$	By Maximum Likelihood (ML) optimization, $\hat{\theta} = \arg\max_{\theta} p(D \mid \theta)$
3. Prediction	$p(x \mid D) = \int p(x \mid \theta) p(\theta \mid D)\, \mathrm{d}\theta$	$p(x \mid D) = p(x \mid \hat{\theta})$

Report Card on Maximum Likelihood Estimation

- Maximum Likelihood (ML) is MAP with uniform prior. MAP is sometimes called a 'penalized' ML procedure:

$$\hat{\theta}_{map} = \arg\max_{\theta} \{ \underbrace{\log p(D \mid \theta)}_{\text{log-likelihood}} + \underbrace{\log p(\theta)}_{\text{penalty}} \}$$

- (good!). ML works rather well if we have a lot of data because the influence of the prior diminishes with more data.

- (good!). Computationally often do-able. Useful fact that makes the optimization easier (since log is monotonously increasing):

$$\arg\max_{\theta} \log p(D|\theta) = \arg\max_{\theta} p(D|\theta)$$

- (bad). Cannot be used for model comparison! When doing ML estimation, the Bayesian model evidence evalutes to zero because the prior probability mass under the likelihood function goes to zero. Therefore, Bayesian model evidence cannot be used to evaluate model performance:

$$p(D|m) = \int p(D|\theta) \cdot p(\theta|m)\,\mathrm{d}\theta$$

$$\begin{aligned}
\text{Bayesian evidence} &= \lim_{(b-a)\to\infty} \int p(D|\theta) \cdot \text{Uniform}(\theta|a,b)\,\mathrm{d}\theta \\
&= \lim_{(b-a)\to\infty} \frac{1}{b-a} \int_a^b p(D|\theta)\,\mathrm{d}\theta \\
&\qquad\qquad < \infty \\
&= 0
\end{aligned}$$

⇒ ML estimation is an approximation to Bayesian learning, but for good reason a very popular learning method when faced with lots of available data.

OPTIONAL SLIDES

Bayesian Evidence as a Model Performance Criterion

- Consider a model $p(x, \theta | m)$ and a data set $D = \{x_1, x_2, \ldots, x_N\}$.
- Given the data set D, the log-evidence for model m decomposes as follows:

$$\begin{aligned}
\log p(D|m) &= \log \frac{p(D|\theta, m)p(\theta|m)}{p(\theta|D, m)} \qquad \text{(use Bayes rule)} \\
&= \log \frac{p(D|\theta, m)p(\theta|m)}{p(\theta|D, m)} \cdot \int p(\theta|D, m)\mathrm{d}\theta \\
&= \log p(D|m) \\
&= \int p(\theta|D, m) \log \frac{p(D|\theta, m)p(\theta|m)}{p(\theta|D, m)} \mathrm{d}\theta \\
&= \int p(\theta|D, m) \log p(D|\theta, m)\mathrm{d}\theta - \int p(\theta|D, m) \log \frac{p(\theta|D, m)}{p(\theta|m)} \mathrm{d}\theta \\
&\qquad \text{data fit} \qquad\qquad\qquad\qquad \text{complexity}
\end{aligned}$$

- The first term (data fit) measures how well the model predicts the data D, after having learned from the data. We want this term to be large (although only focussing on this term could lead to overfitting).
- The second term (complexity) quantifies the amount of information that the model absorbed through learning, i.e., by moving parameter beliefs from $p(\theta|m)$ to $p(\theta|D, m)$.
  - To see this, note that the mutual information between two variables θ and D is defined as

$$I[\theta; D] = \iint p(\theta, D) \log \frac{p(\theta|D)}{p(\theta)} \mathrm{d}\theta \mathrm{d}D$$

- The complexity term regularizes the Bayesian learning process automatically. If you prefer models with high Bayesian evidence, then you prefer models that get a good data fit without need to learn much from the data set. These types of models are said to generalize well, since they can be applied to different data sets without specific adaptations for each data set.
- ⇒ Bayesian learning automatically leads to models that generalize well.

Factor Graphs

Preliminaries

- Goal
  - Introduction to Forney-style factor graphs and message passing-based inference
- Materials
  - Mandatory
    - These lecture notes
    - Loeliger (2007), The factor graph approach to model based signal processing, pp. 1295-1302 (until section V)
  - Optional
    - Frederico Wadehn (2015), Probabilistic graphical models: Factor graphs and more video lecture (highly recommended)
  - References
    - Forney (2001), Codes on graphs: normal realizations

Why Factor Graphs?

- A probabilistic inference task gets its computational load mainly through the need for marginalization (i.e., computing integrals). E.g., for a model $p(x_1,x_2,x_3,x_4,x_5)$, the inference task $p(x_2|x_3)$ is given by

$$p(x_2|x_3) = \frac{p(x_2,x_3)}{p(x_3)} = \frac{\int\cdots\int p(x_1,x_2,x_3,x_4,x_5)\,\mathrm{d}x_1\mathrm{d}x_4\mathrm{d}x_5}{\int\cdots\int p(x_1,x_2,x_3,x_4,x_5)\,\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_4\mathrm{d}x_5}$$

- Since these computations (integrals or sums) suffer from the "curse of dimensionality", we often need to solve a simpler problem in order to get an answer.

- Factor graphs provide a computationally efficient approach to solving inference problems if the generative distribution can be factorized.

- Factorization helps. For instance, if $p(x_1,x_2,x_3,x_4,x_5) = p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)$, then

$$p(x_2|x_3) = \frac{\int\cdots\int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)\,\mathrm{d}x_1\mathrm{d}x_4\mathrm{d}x_5}{\int\cdots\int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)\,\mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_4\mathrm{d}x_5} = \frac{p(x_2,x_3)}{\int p(x_2,x_3)\mathrm{d}x_2}$$

which is computationally much cheaper than the general case above.

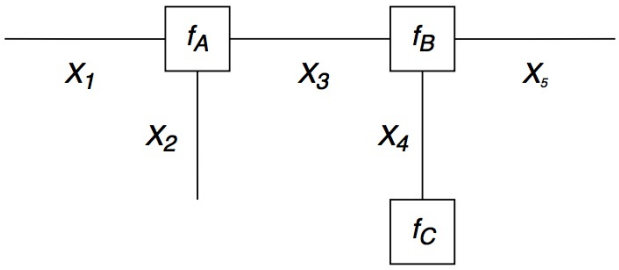
- In this lesson, we discuss how computationally efficient inference in factorized probability distributions can be automated by message passing-based inference in factor graphs.

Factor Graph Construction Rules

- Consider a function

$$f(x_1,x_2,x_3,x_4,x_5) = f_a(x_1,x_2,x_3) \cdot f_b(x_3,x_4,x_5) \cdot f_c(x_4)$$

- The factorization of this function can be graphically represented by a Forney-style Factor Graph (FFG):



- An FFG is an undirected graph subject to the following construction rules (Forney, 2001)
  1. A node for every factor;
  2. An edge (or half-edge) for every variable;
  3. Node g is connected to edge x iff variable x appears in factor g.

Some FFG Terminology

- f is called the global function and $f_.$ are the factors.
- A configuration is an assigment of values to all variables.
- The configuration space is the set of all configurations, i.e., the domain of f
- A configuration $\omega = (x_1,x_2,x_3,x_4,x_5)$ is said to be valid iff $f(\omega) \neq 0$

Equality Nodes for Branching Points

- Note that a variable can appear in maximally two factors in an FFG (since an edge has only two end points).

- Consider the factorization (where x_2 appears in three factors)

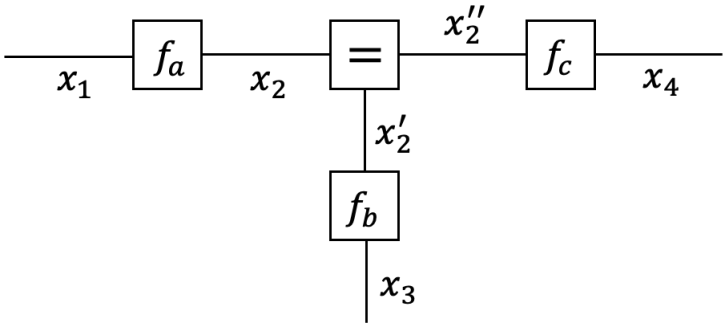
$$f(x_1,x_2,x_3,x_4) = f_a(x_1,x_2) \cdot f_b(x_2,x_3) \cdot f_c(x_2,x_4)$$

- For the factor graph representation, we will instead consider the function g, defined as

$$g(x_1, x_2, x_2', x_2'', x_3, x_4) = f_a(x_1, x_2) \cdot f_b(x_2', x_3) \cdot f_c(x_2'', x_4) \cdot f_=(x_2, x_2', x_2'')$$

  where

$$f_=(x_2, x_2', x_2'') \triangleq \delta(x_2 - x_2')\,\delta(x_2 - x_2'')$$



Equality Nodes for Branching Points, cont'd

- Note that through introduction of auxiliary variables X_2' and X_2'' and a factor $f_=(x_2, x_2', x_2'')$ each variable in g appears in maximally two factors.

- The constraint $f_=(x, x', x'')$ enforces that $X = X' = X''$ for every valid configuration.

- ⇒ Any factorization of a global function f can be represented by a Forney-style Factor Graph.

- Since f is a marginal of g, i.e.,

$$f(x_1, x_2, x_3, x_4) = \iint g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''$$

  it follows that any inference problem on f can be executed by a corresponding inference problem on g, e.g.,

$$\begin{aligned} f(x_1 \mid x_2) &\triangleq \frac{\iint f(x_1, x_2, x_3, x_4)\,\mathrm{d}x_3\mathrm{d}x_4}{\int\cdots\int f(x_1, x_2, x_3, x_4)\,\mathrm{d}x_1\mathrm{d}x_3\mathrm{d}x_4} \\ &= \frac{\int\cdots\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4}{\int\cdots\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_1\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4} \\ &= g(x_1 \mid x_2) \end{aligned}$$

Probabilistic Models as Factor Graphs

- FFGs can be used to express conditional independence (factorization) in probabilistic models.

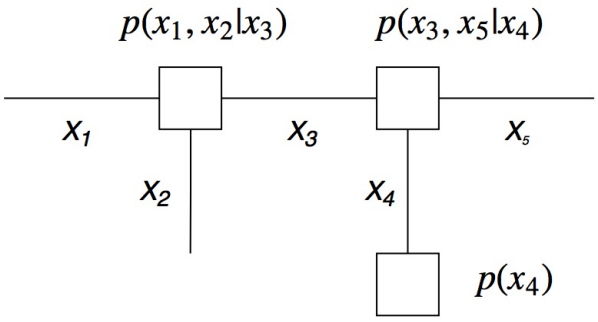
- For example, the (previously shown) graph for $f_a(x_1,x_2,x_3) \cdot f_b(x_3,x_4,x_5) \cdot f_c(x_4)$ could represent the probabilistic model

$$p(x_1,x_2,x_3,x_4,x_5) = p(x_1,x_2|x_3) \cdot p(x_3,x_5|x_4) \cdot p(x_4)$$

  where we identify

$$f_a(x_1,x_2,x_3) = p(x_1,x_2|x_3)$$
$$f_b(x_3,x_4,x_5) = p(x_3,x_5|x_4)$$
$$f_c(x_4) = p(x_4)$$

- This is the graph



Inference by Closing Boxes

- As demonstrated above, factorizations provide opportunities to cut on the amount of needed computations when doing inference. In what follows, we will use FFGs to process these opportunities in an automatic way by message passing between the nodes of the graph.

- Message passing in factor graphs takes advantage of the distributive law of multiplication, which rewrites a sum-of-products as a product-of-sums:

$$ac + ad + bc + cd = (a+b)(c+d)\,.$$

  The LHS consumes 4 multiplications and 3 additions. The RHS consumes 1 multiplication and 2 additions. Inference by message passing implicitly realizes these type of computations by always executing the RHS equations.

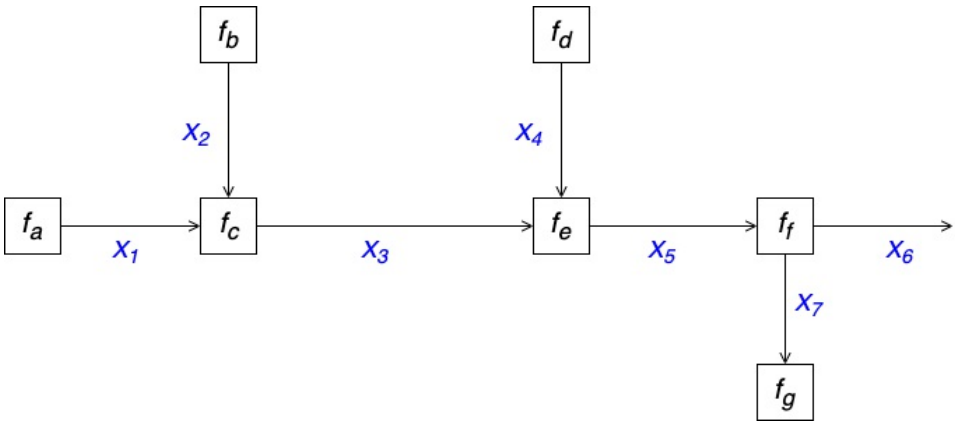
- Assume we wish to compute the marginal

$$\bar{f}(x_3) \triangleq \sum_{x_1, x_2, x_4, x_5, x_6, x_7} f(x_1, x_2, \ldots, x_7)$$

for a model f with given factorization

$$f(x_1, x_2, \ldots, x_7) = f_a(x_1) f_b(x_2) f_c(x_1, x_2, x_3) f_d(x_4) f_e(x_3, x_4, x_5) f_f(x_5, x_6, x_7) f_g(x_7)$$

- The FFG for f is (we will discuss the usage of directed edges below):



- Due to the factorization and the distributive law, we can decompose this sum-of-products to the following product-of-sums:

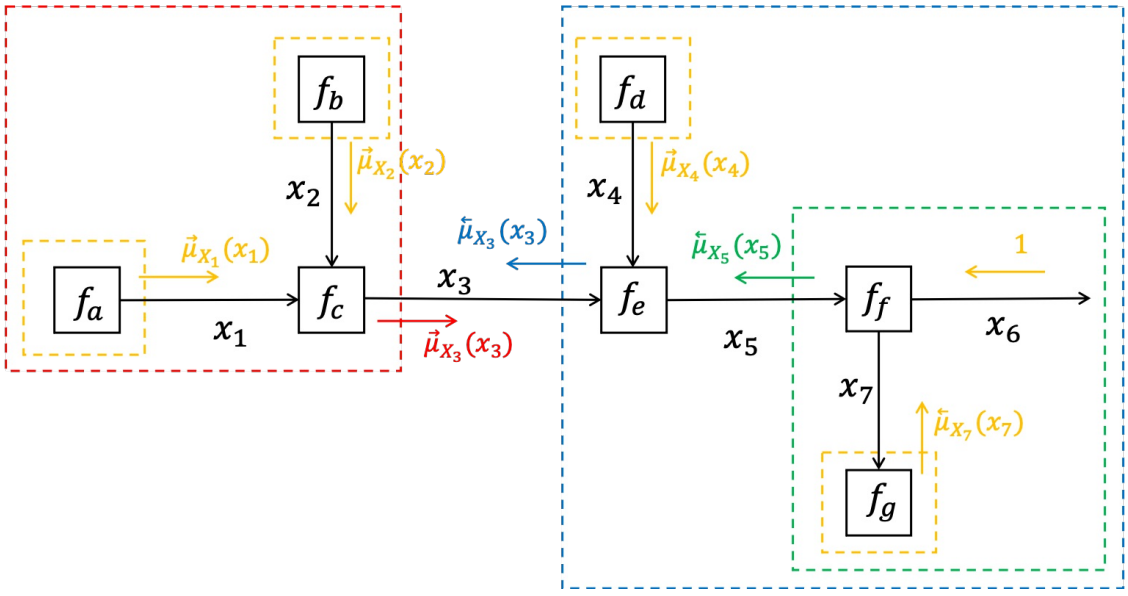
$$\bar{f}(x_3) = \Big(\sum_{x_1, x_2} f_a(x_1)\ f_b(x_2)\ f_c(x_1, x_2, x_3) \Big) \cdot \Big(\sum_{x_4, x_5} f_d(x_4)\ f_e(x_3, x_4, x_5) \cdot \big(\sum_{x_6, x_7} f_f(x_5, x_6, x_7)\ f_g(x_7) \big) \Big)$$

$$\vec{\mu}_{X_1}(x_1)\ \vec{\mu}_{X_2}(x_2) \qquad \vec{\mu}_{X_4}(x_4) \qquad \overleftarrow{\mu}_{X_7}(x_7)$$

$$\vec{\mu}_{X_3}(x_3) \qquad \overleftarrow{\mu}_{X_5}(x_5)$$

$$\overleftarrow{\mu}_{X_3}(x_3)$$

which is computationally (much) lighter than executing the full sum $\sum_{x_1, \ldots, x_7} f(x_1, x_2, \ldots, x_7)$



- Note that the new factor $\vec{\mu}_{X_3}(x_3)$ is obtained by multiplying all enclosed factors (f_a, f_b, f_c) by the red dashed box, followed by marginalization (summing) over all enclosed variables (x_1, x_2).

- This is the Closing the Box-rule, which is a general recipe for marginalization of latent variables (inside the box) and leads to a new factor that has the variables (edges) that cross the box as arguments. For instance, the argument of the remaining factor $\vec{\mu}_{X_3}(x_3)$ is the variable on the edge that crosses the red box (x_3).

- Hence, $\vec{\mu}_{X_3}(x_3)$ can be interpreted as a message from the red box toward variable x_3.

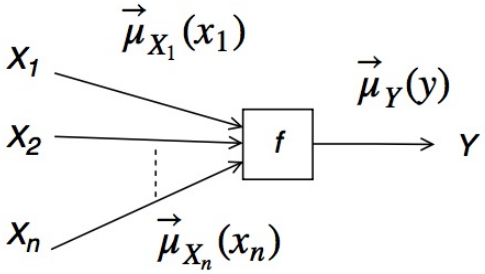
- We drew directed edges in the FFG in order to distinguish forward messages $\vec{\mu}_{\cdot}(\cdot)$ (in the same direction as the arrow of the edge) from backward messages $\overleftarrow{\mu}_{\cdot}(\cdot)$ (in opposite direction). This is just a notational convenience since an FFG is computationally an undirected graph.

Sum-Product Algorithm

- This recursive pattern for computing and passing messages applies generally and is called the Sum-Product update rule, which is really just a special case of the closing-the-box rule: For any node, the outgoing message is obtained by taking the product of all incoming messages and the node function, followed by summing out (marginalization) all incoming variables. What is left (the outgoing message) is a function of the outgoing variable only (Loeliger (2007), pg.1299):

$$\vec{\mu}_Y(y) = \sum_{x_1, \ldots, x_n} \vec{\mu}_{X_1}(x_1) \cdots \vec{\mu}_{X_n}(x_n) \cdot f(y, x_1, \ldots, x_n)$$

outgoing message | incoming messages | node function



- Note that all message update rules can be computed from information that is locally available at each node.

- If the factor graph for a function f has no cycles (i.e., the graph is a tree), then the marginal $\bar{f}(x_3) = \sum_{x_1, x_2, x_4, x_5, x_6, x_7} f(x_1, x_2, \ldots, x_7)$ is given by the Sum-Product Theorem:

$$\bar{f}(x_3) = \vec{\mu}_{X_3}(x_3) \cdot \overleftarrow{\mu}_{X_3}(x_3)$$

- It follows that the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1,x_2,\ldots,x_7)$ can be efficiently computed through sum-product messages. Executing inference through SP message passing is called the Sum-Product Algorithm.

- In a tree graph, start with messages from the terminals and keep passing messages through the internal nodes towards the "target" variable (x_3 in above problem) until you have both the forward and backward message for the target variable.

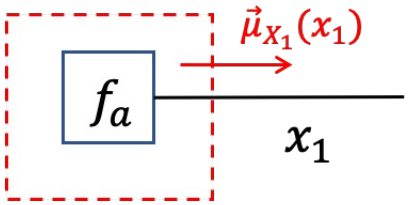
- In a tree graph, if you continue to pass messages throughout the graph, the Sum-Product Algorithm computes exact marginals for all hidden variables.
- If the graph contains cycles, we have in principle an infinite tree by "unrolling" the graph. In this case, the SP Algorithm is not guaranteed to find exact marginals. In practice, if we apply the SP algorithm for just a few iterations ("unrolls"), then we often find satisfying approximate marginals.

Terminal Nodes and Processing Observations

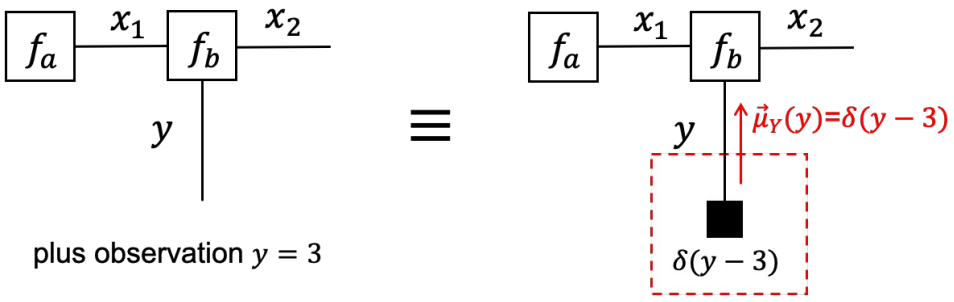
- The message out of a terminal node (attached to only 1 edge) is the factor itself. For instance, closing a box around terminal node $f_a(x_1)$ would lead to

$$\vec{\mu}_{X_1}(x_1) \triangleq \sum_{\substack{\text{enclosed} \\ \text{variables}}} \prod_{\substack{\text{enclosed} \\ \text{factors}}} f_a(x_1) = f_a(x_1)$$

  since there are no enclosed variables.



- The message from a half-edge is 1 (one). You can verify this by imagining that a half-edge x can be terminated by a node function $f(x) = 1$ without affecting any inference issue.
- We can use terminal nodes represent observations, e.g., add a factor $f(y) = \delta(y-3)$ to terminate the half-edge for variable Y if $y = 3$ is observed.



- Terminal nodes that carry observations are denoted by small black boxes.

Automating Bayesian Inference by Message Passing

- The foregoing message update rules can be worked out in closed-form and put into tables (e.g., see Tables 1 through 6 in Loeliger (2007) for many standard factors such as essential probability distributions and operations such as additions, fixed-gain multiplications and branching (equality nodes).
- In the optional slides below, we have worked out a few of these update rules, eg, for the equality node, the addition node and the multiplication node.
- If the update rules for all node types in a graph have been tabulated, then inference by message passing comes down to executing a set of table-lookup operations, thus creating a completely automatable Bayesian inference framework.
- In our research lab BIASlab (FLUX 7.060), we are developing ForneyLab, which is a (Julia) toolbox for automating Bayesian inference by message passing in a factor graph.

Example: Bayesian Linear Regression by Message Passing

- Recall: the goal of regression is to estimate an unknown function from a set of (noisy) function values.

- Assume we want to estimate some function $f: \mathrm{R}^D \rightarrow \mathrm{R}$ from data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $y_i = f(x_i) + \epsilon_i$.

model specification

- We will assume a linear model with white Gaussian noise and a Gaussian prior on the coefficients w:

$$\begin{aligned} y_i &= w^T x_i + \epsilon_i \\ \epsilon_i &\sim \mathcal{N}(0, \sigma^2) \\ w &\sim \mathcal{N}(0, \Sigma) \end{aligned}$$

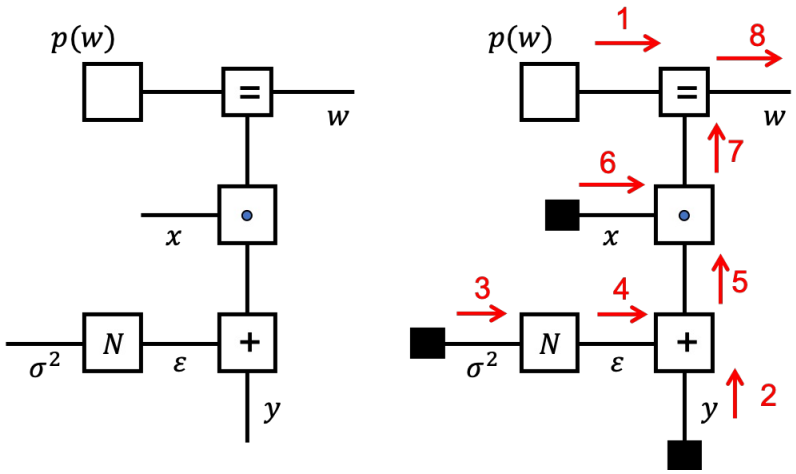
  or equivalently

$$\begin{aligned} p(w, \epsilon, D) &= \overbrace{p(w)}^{\text{weight prior}} \overbrace{\prod_{i=1}^{N} p(y_i \mid x_i, w, \epsilon_i)}^{\text{regression model}} \overbrace{p(\epsilon_i)}^{\text{noise model}} \\ &= \mathcal{N}(w \mid 0, \Sigma) \prod_{i=1}^{N} \delta(y_i - w^T x_i - \epsilon_i) \; \mathcal{N}(\epsilon_i \mid 0, \sigma^2) \end{aligned}$$

Inference (parameter estimation)

- We are interested in inferring the posterior $p(w \mid D)$. We will execute inference by message passing on the FFG for the model.

- The left figure shows the factor graph for this model.
- The right figure shows the message passing scheme.

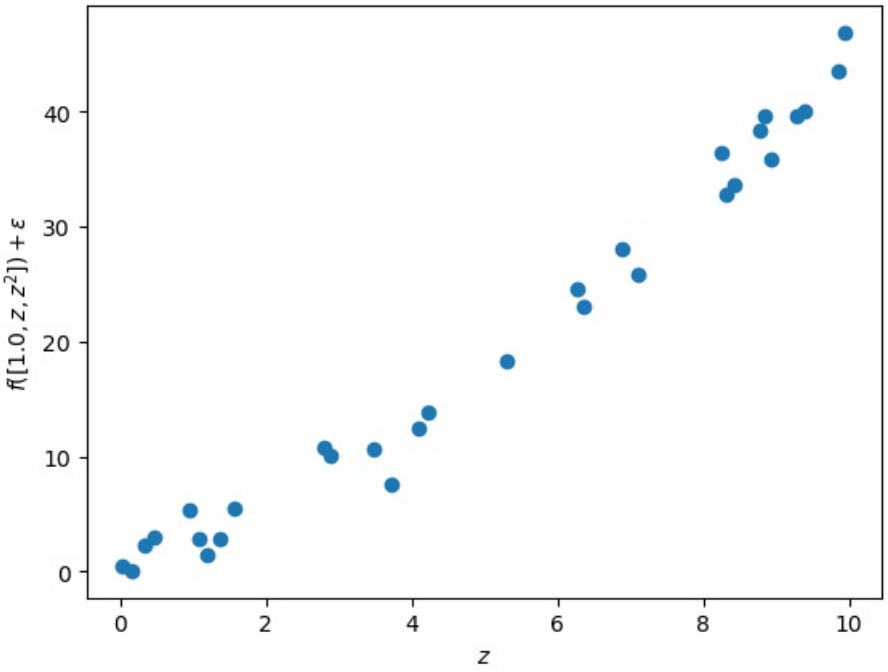


CODE EXAMPLE

Let's solve this problem by message passing-based inference with Julia's FFG toolbox ForneyLab.

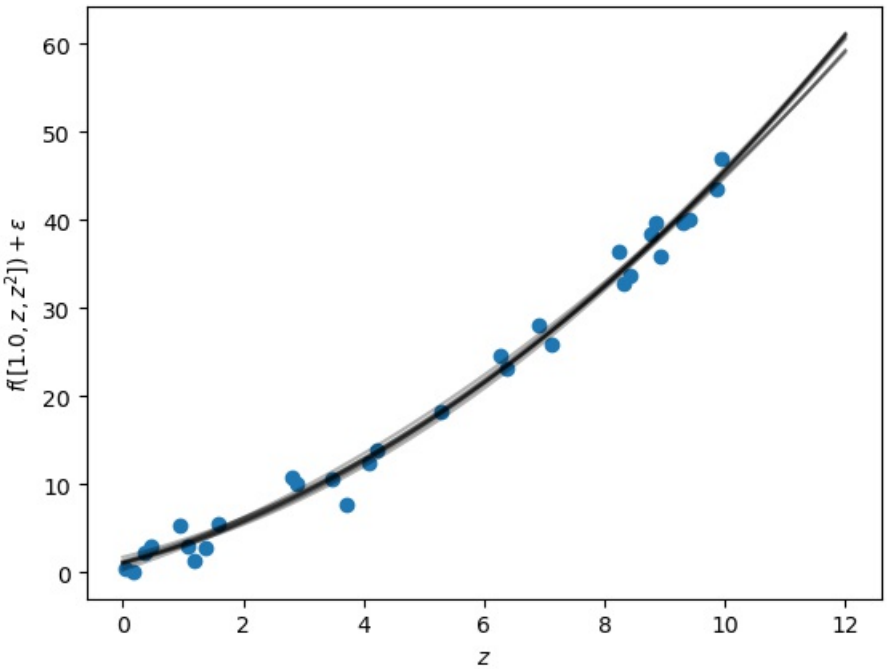
```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate();
IJulia.clear_output();
```

```
using PyPlot, ForneyLab, LinearAlgebra

# Parameters
Σ = 1e5 * Diagonal(I,3) # Covariance matrix of prior on w
σ2 = 2.0          # Noise variance

# Generate data set
w = [1.0; 2.0; 0.25]
N = 30
z = 10.0*rand(N)
x_train = [[1.0; z; z^2] for z in z] # Feature vector x = [1.0; z; z^2]
f(x) = (w'*x)[1]
y_train = map(f, x_train) + sqrt(σ2)*randn(N) # y[i] = w' * x[i] + ϵ
scatter(z, y_train); xlabel(L"z"); ylabel(L"f([1.0, z, z^2]) + \epsilon");
```



Now build the factor graph in ForneyLab, perform sum-product message passing and plot results (mean of posterior).

```
# Build factorgraph

fg = FactorGraph()
@RV w ~ GaussianMeanVariance(constant(zeros(3)), constant(Σ, id=:Σ), id=:w) # p(w)
for t=1:N
    x_t = Variable(id=:x_*t)
    d_t = Variable(id=:d_*t) # d=w'*x
    DotProduct(d_t, x_t, w) # p(f|w,x)
    @RV y_t ~ GaussianMeanVariance(d_t, constant(σ2, id=:σ2_*t), id=:y_*t) # p(y|d)
    placeholder(x_t, :x, index=t, dims=(3,))
    placeholder(y_t, :y, index=t);
end

# Build and run message passing algorithm
algo = messagePassingAlgorithm(w)
source_code = algorithmSourceCode(algo)
eval(Meta.parse(source_code))
data = Dict(:x => x_train, :y => y_train)
w_posterior_dist = step!(data)[:w]

# Plot result
println("Posterior distribution of w: $(w_posterior_dist)")
scatter(z, y_train); xlabel(L"z"); ylabel(L"f([1.0, z, z^2]) + \epsilon");
z_test = collect(0:0.2:12)
x_test = [[1.0; z; z^2] for z in z_test]
for sample=1:10
    w = ForneyLab.sample(w_posterior_dist)
    f_est(x) = (w'*x)[1]
    plot(z_test, map(f_est, x_test), "k-", alpha=0.3);
end
```

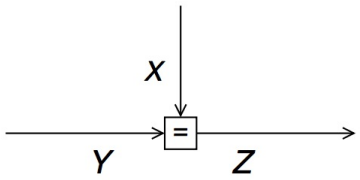



```
Posterior distribution of w: 𝒩(xi=[2.98e+02, 2.27e+03, 1.89e+04], w=[[15.00, 75.19, 5.53e+0
2][75.19, 5.53e+02, 4.52e+03][5.53e+02, 4.52e+03, 3.87e+04]])
```

OPTIONAL SLIDES

Sum-Product Messages for the Equality Node

- Let's compute the SP messages for the equality node $f_=(x,y,z) = \delta(z-x)\delta(z-y)$:



$$\begin{aligned}\vec{\mu}_Z(z) &= \iint \vec{\mu}_X(x)\vec{\mu}_Y(y)\,\delta(z-x)\delta(z-y)\,\mathrm{d}x\mathrm{d}y \\ &= \vec{\mu}_X(z)\int \vec{\mu}_Y(y)\,\delta(z-y)\,\mathrm{d}y \\ &= \vec{\mu}_X(z)\vec{\mu}_Y(z)\end{aligned}$$

- By symmetry, this also implies (for the same equality node) that

$$\overleftarrow{\mu}_X(x) = \vec{\mu}_Y(x)\,\overleftarrow{\mu}_Z(x) \quad \text{and}$$
$$\overleftarrow{\mu}_Y(y) = \vec{\mu}_X(y)\,\overleftarrow{\mu}_Z(y)\,.$$

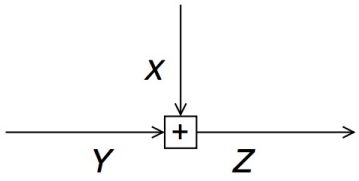
- Let us now consider the case of Gaussian messages $\vec{\mu}_X(x) = \mathcal{N}(m_X, V_X)$, $\vec{\mu}_Y(y) = \mathcal{N}(m_Y, V_Y)$ and $\vec{\mu}_Z(z) = \mathcal{N}(m_Z, V_Z)$. Let's also define the precision matrices $W_X \triangleq V_X^{-1}$ and similarly for Y and Z. Then applying the SP update rule leads to multiplication of two Gaussian distributions, resulting in

$$\begin{aligned}W_Z &= W_X + W_Y \\ W_Z m_z &= W_X m_X + W_Y m_Y\end{aligned}$$

- It follows that message passing through an equality node is similar to applying Bayes rule, i.e., fusion of two information sources. Does this make sense?

Sum-Product Messages for the Addition Node

- Next, consider an addition node $f_+(x,y,z) = \delta(z-x-y)$:



$$\begin{aligned}\vec{\mu}_Z(z) &= \iint \vec{\mu}_X(x)\vec{\mu}_Y(y)\,\delta(z-x-y)\,\mathrm{d}x\mathrm{d}y \\ &= \int \vec{\mu}_X(x)\vec{\mu}_Y(z-x)\,\mathrm{d}x\,,\end{aligned}$$

i.e., $\vec{\mu}_Z$ is the convolution of the messages $\vec{\mu}_X$ and $\vec{\mu}_Y$.

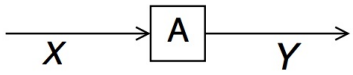
- Of course, for Gaussian messages, these update rules evaluate to

$$m_Z = m_X + m_Y, \text{ and } V_z = V_X + V_Y.$$

- Exercise: For the same summation node, work out the SP update rule for the backward message $\overleftarrow{\mu}_X(x)$ as a function of $\vec{\mu}_Y(y)$ and $\overleftarrow{\mu}_Z(z)$. And further refine the answer for Gaussian messages.

Sum-Product Messages for Multiplication Nodes

- Next, let us consider a multiplication by a fixed (invertible matrix) gain $f_A(x,y) = \delta(y - Ax)$



$$\vec{\mu}_Y(y) = \int \vec{\mu}_X(x)\,\delta(y - Ax)\,\mathrm{d}x = \vec{\mu}_X(A^{-1}y)\,.$$

- For a Gaussian message input message $\vec{\mu}_X(x) = \quad (m_X, V_X)$, the output message is also Gaussian with

$$m_Y = Am_X, \text{ and } V_Y = AV_XA^T$$

since

$$\begin{aligned}\vec{\mu}_Y(y) &= \vec{\mu}_X(A^{-1}y) \\ &\propto \exp\left(-\frac{1}{2}\left(A^{-1}y - m_X\right)^T V_X^{-1}\left(A^{-1}y - m_X\right)\right) \\ &= \exp\left(-\frac{1}{2}\left(y - Am_X\right)^T A^{-T}V_X^{-1}A\left(y - Am_X\right)\right) \\ &\propto \quad (Am_X, AV_XA^T)\,.\end{aligned}$$

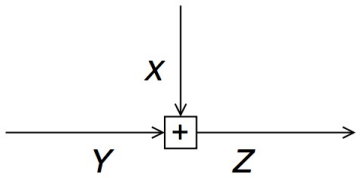
- Excercise: Proof that, for the same factor $\delta(y - Ax)$ and Gaussian messages, the (backward) sum-product message $\overleftarrow{\mu}_X$ is given by

$$\begin{aligned}\overleftarrow{\xi}_X &= A^T\overleftarrow{\xi}_Y \\ \overleftarrow{W}_X &= A^T\overleftarrow{W}_YA\end{aligned}$$

where $\overleftarrow{\xi}_X \triangleq \overleftarrow{W}_X\overleftarrow{m}_X$ and $\overleftarrow{W}_X \triangleq \overleftarrow{V}_X^{-1}$ (and similarly for Y).

CODE EXAMPLE

Let's calculate the Gaussian forward and backward messages for the addition node in ForneyLab.




```
# Forward message towards Z
fg = FactorGraph()
@RV x ~ GaussianMeanVariance(constant(1.0), constant(1.0), id=:x)
@RV y ~ GaussianMeanVariance(constant(2.0), constant(1.0), id=:y)
@RV z = x + y; z.id = :z

q = PosteriorFactorization(fg)

algo1 = messagePassingAlgorithm(z, id=:_forward_Z)
source_code1 = algorithmSourceCode(algo1)
eval(Meta.parse(source_code1))
msg_forward_Z = step_forward_Z!(Dict())[:z]
print("Forward message on Z: $(msg_forward_Z)")
```

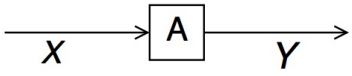
```
Forward message on Z: 𝒩(m=3.00, v=2.00)
```

```
# Backward message towards X
fg = FactorGraph()
@RV x; x.id=:x
@RV y ~ GaussianMeanVariance(constant(2.0), constant(1.0), id=:y)
@RV z = x + y
GaussianMeanVariance(z, constant(3.0), constant(1.0), id=:z)

q = PosteriorFactorization(fg)

algo2 = messagePassingAlgorithm(x, id=:_backward_X)
source_code2 = algorithmSourceCode(algo2)
eval(Meta.parse(source_code2))
msg_backward_X = step_backward_X!(Dict())[:x]
print("Backward message on X: $(msg_backward_X)")
```

```
Backward message on X: 𝒩(m=1.00, v=2.00)
```

CODE EXAMPLE

In the same way we can also investigate the forward and backward messages for the matrix multiplication ("gain") node



```
# Forward message towards Y
fg = FactorGraph()
@RV x ~ GaussianMeanVariance(1.0, 1.0)
@RV y = 4.0 * x

q = PosteriorFactorization(fg)
#This is where the bugs live..
algo3 = messagePassingAlgorithm(y, id=:_y_fwd)
source_code3 = algorithmSourceCode(algo3)
eval(Meta.parse(source_code3))
msg_forward_Y = step_y_fwd!(Dict())[:y]
print("Forward message on Y: $(msg_forward_Y)")
```

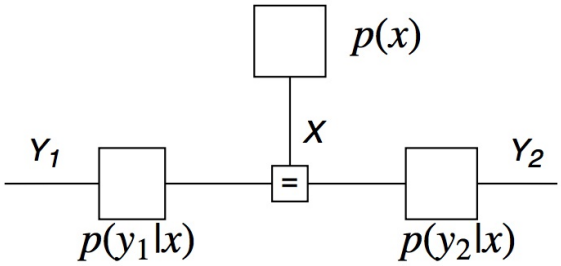
```
Forward message on Y: 𝒩(m=4.00, v=16.00)
```

Example

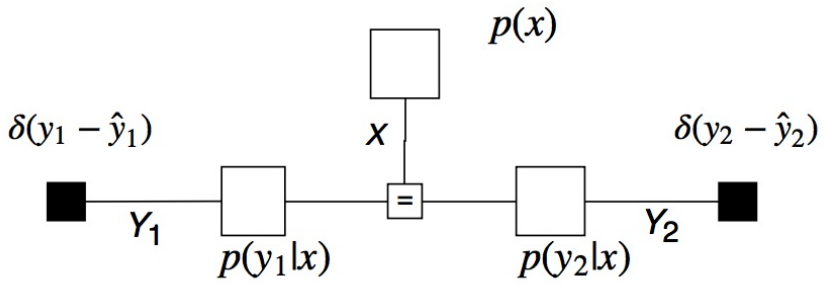
- Consider a generative model

$$p(x, y_1, y_2) = p(x)p(y_1 \mid x)p(y_2 \mid x).$$

  - This model expresses the assumption that Y_1 and Y_2 are independent measurements of X.



- Assume that we are interested in the posterior for X after observing $Y_1 = \hat{y}_1$ and $Y_2 = \hat{y}_2$. The posterior for X can be inferred by applying the sum-product algorithm to the following graph:



- (Note that) we usually draw terminal nodes for observed variables in the graph by smaller solid-black squares. This is just to help the visualization of the graph, since the computational rules are no different than for other nodes.

CODE EXAMPLE

We'll use ForneyLab, a factor graph toolbox for Julia, to build the above graph, and perform sum-product message passing to infer the posterior $p(x|y_1,y_2)$. We assume $p(y_1|x)$ and $p(y_2|x)$ to be Gaussian likelihoods with known variances:

$$\begin{aligned} p(y_1 \mid x) &= \quad (y_1 \mid x, v_{y1}) \\ p(y_2 \mid x) &= \quad (y_2 \mid x, v_{y2}) \end{aligned}$$

Under this model, the posterior is given by:

$$\begin{aligned} p(x \mid y_1,y_2) &\propto \overbrace{p(y_1 \mid x)p(y_2 \mid x)}^{\text{likelihood}}\overbrace{p(x)}^{\text{prior}} \\ &= \quad (x \mid \hat{y}_1, v_{y1}) \quad (x \mid \hat{y}_2, v_{y2}) \quad (x \mid m_x, v_x) \end{aligned}$$

so we can validate the answer by solving the Gaussian multiplication manually.

```
using ForneyLab

# Data
y1_hat = 1.0
y2_hat = 2.0

# Construct the factor graph
fg = FactorGraph()
@RV x  ~ GaussianMeanVariance(constant(0.0), constant(4.0), id=:x) # Node p(x)
@RV y1 ~ GaussianMeanVariance(x, constant(1.0))                    # Node p(y1|x)
@RV y2 ~ GaussianMeanVariance(x, constant(2.0))                    # Node p(y2|x)
Clamp(y1, y1_hat)                   # Terminal (clamp) node for y1
Clamp(y2, y2_hat)                   # Terminal (clamp) node for y2
# draw(fg) # draw the constructed factor graph

# Perform sum-product message passing
algo4 = messagePassingAlgorithm(x, id=:_x)
source_code4 = algorithmSourceCode(algo4)
eval(Meta.parse(source_code4))
x_marginal = step_x!(Dict())[:x]
println("Sum-product message passing result: p(x|y1,y2) = 𝒩($(mean(x_marginal)),$(var(x_marginal)
))")

# Calculate mean and variance of p(x|y1,y2) manually by multiplying 3 Gaussians (see lesson 4 for
details)
v = 1 / (1/4 + 1/1 + 1/2)
m = v * (0/4 + y1_hat/1.0 + y2_hat/2.0)
println("Manual result: p(x|y1,y2) = 𝒩($(m), $(v))")
```

```
Sum-product message passing result: p(x|y1,y2) = 𝒩(1.1428571428571428,0.5714285714285714)
Manual result: p(x|y1,y2) = 𝒩(1.1428571428571428, 0.5714285714285714)
```

```
# Backward message towards X
fg = FactorGraph()
x = Variable(id=:x)
@RV y = constant(4.0) * x
GaussianMeanVariance(y, constant(2.0), constant(1.0))

q = PosteriorFactorization(fg)

algo5 = messagePassingAlgorithm(x, id=:_x_fwd2)
source_code5 = algorithmSourceCode(algo5)
eval(Meta.parse(source_code5))
msg_backward_X = step_x_fwd2!(Dict())[:x]
print("Backward message on X: $(msg_backward_X)")
```

```
Backward message on X: 𝒩(xi=8.00, w=16.00)
```

Continuous Data and the Gaussian Distribution

Preliminaries

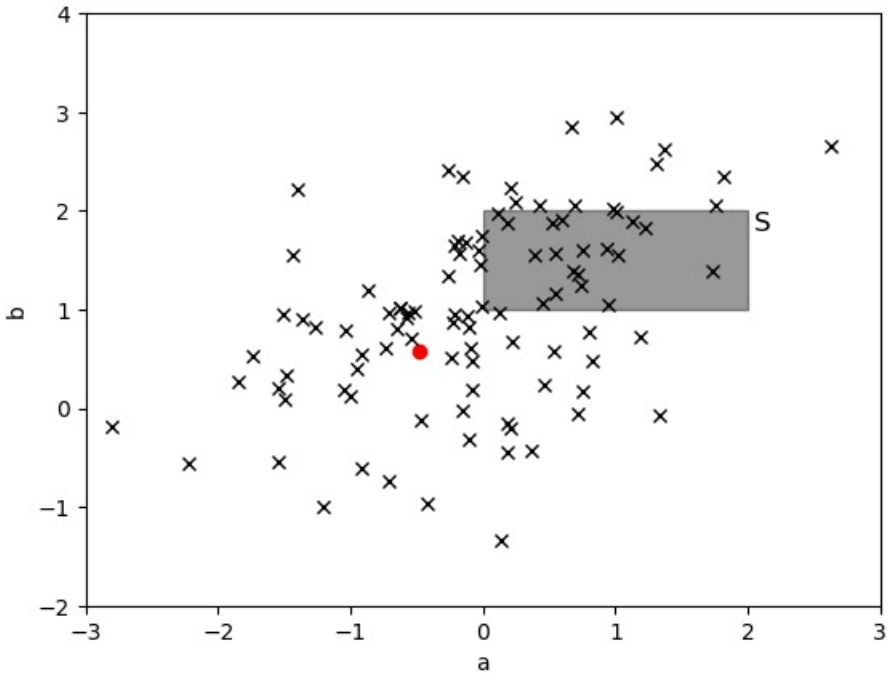
- Goal
  - Review of information processing with Gaussian distributions in linear systems
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 85-93
    - MacKay - 2006 - The Humble Gaussian Distribution (highly recommended!)
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.30-34, section 2.8, the Gaussian distribution

Example Problem

Consider a set of observations $D = \{x_1, \ldots, x_N\}$ in the 2-dimensional plane (see Figure). All observations were generated by the same process. We now draw an extra observation $x_\bullet = (a, b)$ from the same data generating process. What is the probability that $x_\bullet$ lies within the shaded rectangle S?

```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate();
IJulia.clear_output();
```

```
using Distributions, PyPlot
N = 100
generative_dist = MvNormal([0,1.], [0.8 0.5; 0.5 1.0])
function plotObservations(obs::Matrix)
    plot(obs[1,:], obs[2,:], "kx", zorder=3)
    fill_between([0., 2.], 1., 2., color="k", alpha=0.4, zorder=2) # Shaded area
    text(2.05, 1.8, "S", fontsize=12)
    xlim([-3,3]); ylim([-2,4]); xlabel("a"); ylabel("b")
end
D = rand(generative_dist, N) # Generate observations from generative_dist
plotObservations(D)
x_dot = rand(generative_dist) # Generate x·
plot(x_dot[1], x_dot[2], "ro");
```



The Gaussian Distribution

- Consider a random (vector) variable $x \in \mathrm{R}^M$ that is "normally" (i.e., Gaussian) distributed. The moment parameterization of the Gaussian distribution is completely specified by its mean μ and variance Σ and given by

$$p(x\,|\mu,\Sigma) = \quad (x\,|\mu,\Sigma) \triangleq \frac{1}{\sqrt{(2\pi)^M|\,\Sigma\,|}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}.$$

  where $|\Sigma| \triangleq \det(\Sigma)$ is the determinant of Σ.

- Alternatively, the _canonical_ (a.k.a. _natural_ or _information_) parameterization of the Gaussian distribution is given by

$$p(x\,|\,\eta,\Lambda) = \quad_c(x\,|\,\eta,\Lambda) = \exp\left\{a + \eta^T x - \frac{1}{2}x^T\Lambda x\right\}.$$

  - $a = -\frac{1}{2}\left(M\log(2\pi) - \log|\,\Lambda\,| + \eta^T\Lambda\eta\right)$ is the normalizing constant that ensures that $\int p(x)\mathrm{d}x = 1$.
  - $\Lambda = \Sigma^{-1}$ is called the precision matrix.
  - $\eta = \Sigma^{-1}\mu$ is the natural mean or for clarity often called the precision-weighted mean.

Why the Gaussian?

- Why is the Gaussian distribution so ubiquitously used in science and engineering? (see Jaynes, pg.220).

- (1) Operations on probability distributions tend lead to Gaussian distributions:
  - Any smooth function with single rounded maximum, if raised to higher and higher powers, goes into a Gaussian function. (Example: sequential Bayesian inference).
  - The Gaussian distribution has higher entropy than any other with the same variance.
    - Therefore any operation on a probability distribution that discards information but preserves variance gets us closer to a Gaussian.
    - As an example, see Jaynes (pg.220) for how this leads to the Central Limit Theorem, which results from performing convolution operations on distributions.

- (2) Once the Gaussian has been attained, this form tends to be preserved. e.g.,
  - The convolution of two Gaussian functions is another Gaussian function (useful in sum of 2 variables and linear transformations)
  - The product of two Gaussian functions is another Gaussian function (useful in Bayes rule).
  - The Fourier transform of a Gaussian function is another Gaussian function.

Transformations and Sums of Gaussian Variables

- A linear transformation $z = Ax + b$ of a Gaussian variable $\quad(x\,|\,\mu_x, \Sigma_x)$ is Gaussian distributed as

$$p(z) = \quad\left(z \mid A\mu_x + b, A\Sigma_x A^T\right)$$

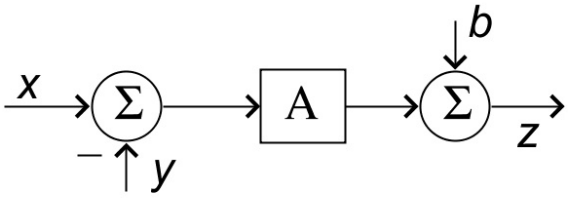
  - In fact, after a linear transformation $z = Ax + b$, no matter how x is distributed, the mean and variance of z are always given by $\mu_z = A\mu_x + b$ and $\Sigma_z = A\Sigma_x A^T$, respectively (see probability theory review lesson). In case x is not Gaussian, higher order moments may be needed to specify the distribution for z.

- The sum of two independent Gaussian variables is also Gaussian distributed. Specifically, if $x \sim \mathcal{N}(\mu_x, \Sigma_x)$ and $y \sim \mathcal{N}(\mu_y, \Sigma_y)$, then the PDF for $z = x + y$ is given by

$$\begin{aligned} p(z) &= \mathcal{N}(x \mid \mu_x, \Sigma_x) * \mathcal{N}(y \mid \mu_y, \Sigma_y) \\ &= \mathcal{N}(z \mid \mu_x + \mu_y, \Sigma_x + \Sigma_y) \end{aligned}$$

  - The sum of two Gaussian distributions is NOT a Gaussian distribution. Why not?

Example: Gaussian Signals in a Linear System



- Given independent variables $x \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and $y \sim \mathcal{N}(\mu_y, \sigma_y^2)$, what is the PDF for $z = A \cdot (x - y) + b$? (see Exercises)

- Think about the role of the Gaussian distribution for stochastic linear systems in relation to what sinusoidals mean for deterministic linear system analysis.

Bayesian Inference for the Gaussian

- Let's estimate a constant θ from one 'noisy' measurement x about that constant.
- We assume the following measurement equations (the tilde $\sim$ means: 'is distributed as'):

$$\begin{aligned} x &= \theta + \epsilon \\ \epsilon &\sim \mathcal{N}(0, \sigma^2) \end{aligned}$$

- Also, let's assume a Gaussian prior for θ

$$\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

Model specification

- Note that you can rewrite these specifications in probabilistic notation as follows:

$$\begin{aligned} p(x \mid \theta) &= \mathcal{N}(x \mid \theta, \sigma^2) \\ p(\theta) &= \mathcal{N}(\theta \mid \mu_0, \sigma_0^2) \end{aligned}$$

- (Notational convention). Note that we write $\epsilon \sim \mathcal{N}(0, \sigma^2)$ but not $\epsilon \sim \mathcal{N}(\epsilon \mid 0, \sigma^2)$, and we write $p(\theta) = \mathcal{N}(\theta \mid \mu_0, \sigma_0^2)$ but not $p(\theta) = \mathcal{N}(\mu_0, \sigma_0^2)$.

Inference

- For simplicity, we assume that the variance σ^2 is given and will proceed to derive a Bayesian posterior for the mean θ. The case for Bayesian inference for σ^2 with a given mean is discussed in the optional slides.

- Let's do Bayes rule for the posterior PDF $p(\theta|x)$.

$$
\begin{aligned}
p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} &\propto p(x|\theta)p(\theta) \\
&= \quad (x|\theta,\sigma^2) \quad (\theta|\mu_0,\sigma_0^2) \\
&\propto \exp\left\{ -\frac{(x-\theta)^2}{2\sigma^2} - \frac{(\theta-\mu_0)^2}{2\sigma_0^2} \right\} \\
&\propto \exp\left\{ \theta^2 \cdot \left(-\frac{1}{2\sigma_0^2} - \frac{1}{2\sigma^2} \right) + \theta \cdot \left(\frac{\mu_0}{\sigma_0^2} + \frac{x}{\sigma^2} \right) \right\} \\
&= \exp\left\{ -\frac{\sigma_0^2+\sigma^2}{2\sigma_0^2\sigma^2} \left(\theta - \frac{\sigma_0^2 x + \sigma^2 \mu_0}{\sigma^2+\sigma_0^2} \right)^2 \right\}
\end{aligned}
$$

  which we recognize as a Gaussian distribution w.r.t. θ.

- (Just as an aside,) this computational 'trick' for multiplying two Gaussians is called completing the square. The procedure makes use of the equality

$$
ax^2 + bx + c_1 = a\left(x + \frac{b}{2a} \right)^2 + c_2
$$

- In particular, it follows that the posterior for θ is

$$
p(\theta|x) = \quad (\theta|\,\mu_1,\sigma_1^2)
$$

  where

$$
\frac{1}{\sigma_1^2} = \frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2} = \frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}
$$

$$
\mu_1 = \frac{\sigma_0^2 x + \sigma^2 \mu_0}{\sigma^2+\sigma_0^2} = \sigma_1^2 \left(\frac{1}{\sigma_0^2}\mu_0 + \frac{1}{\sigma^2} x \right)
$$

(Multivariate) Gaussian Multiplication

- So, multiplication of two Gaussian distributions yields another (unnormalized) Gaussian with
  - posterior precision equals sum of prior precisions
  - posterior precision-weighted mean equals sum of prior precision-weighted means

- As we just saw, great application to Bayesian inference!

$$\underset{\text{posterior}}{\text{Gaussian}} \propto \underset{\text{likelihood}}{\text{Gaussian}} \times \underset{\text{prior}}{\text{Gaussian}}$$

- In general, the multiplication of two multi-variate Gaussians yields an (unnormalized) Gaussian:

$$(x \mid \mu_a, \Sigma_a) \cdot (x \mid \mu_b, \Sigma_b) = \underset{\text{normalization constant}}{(\mu_a \mid \mu_b, \Sigma_a + \Sigma_b)} \cdot (x \mid \mu_c, \Sigma_c)$$

  where

$$\Sigma_c^{-1} = \Sigma_a^{-1} + \Sigma_b^{-1}$$
$$\Sigma_c^{-1}\mu_c = \Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b$$

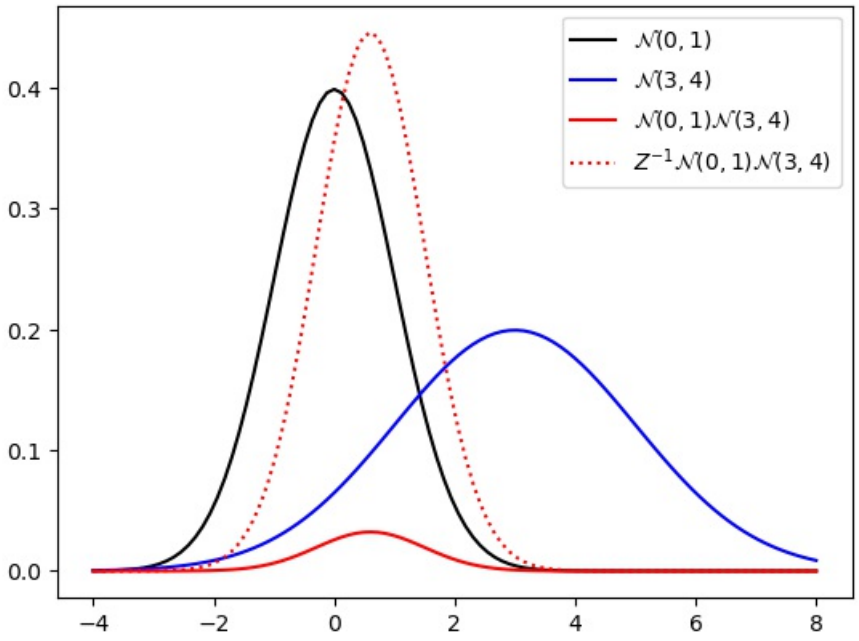
- ⇒ Note that Bayesian inference is trivial in the canonical parameterization of the Gaussian, where we would get

$$\Lambda_c = \Lambda_a + \Lambda_b \quad \text{(precisions add)}$$
$$\eta_c = \eta_a + \eta_b \quad \text{(precision-weighted means add)}$$

CODE EXAMPLE

Let's plot the exact product of two Gaussian PDFs as well as the normalized product according to the above derivation.

```
using PyPlot, Distributions
d1 = Normal(0, 1) # μ=0, σ^2=1
d2 = Normal(3, 2) # μ=3, σ^2=4

# Calculate the parameters of the product d1*d2
s2_prod = (d1.σ^-2 + d2.σ^-2)^-1
m_prod = s2_prod * ((d1.σ^-2)*d1.μ + (d2.σ^-2)*d2.μ)
d_prod = Normal(m_prod, sqrt(s2_prod)) # Note that we neglect the normalization constant.

# Plot stuff
x = range(-4, stop=8, length=100)
plot(x, pdf.(d1,x), "k")
plot(x, pdf.(d2,x), "b")
plot(x, pdf.(d1,x) .* pdf.(d2,x), "r-") # Plot the exact product
plot(x, pdf.(d_prod,x), "r:")           # Plot the normalized Gaussian product
legend([L"\mathcal{N}(0,1)",
        L"\mathcal{N}(3,4)",
        L"\mathcal{N}(0,1) \mathcal{N}(3,4)",
        L"Z^{-1} \mathcal{N}(0,1) \mathcal{N}(3,4)"]);
```



The solid and dotted red curves are identical up to a scaling factor Z.

Bayesian Inference with multiple Observations

- Now consider that we measure a data set $D = \{x_1, x_2, \ldots, x_N\}$, with measurements

$$x_n = \theta + \epsilon_n$$
$$\epsilon_n \sim \quad (0, \sigma^2)$$

  and the same prior for θ:

$$\theta \sim \quad (\mu_0, \sigma_0^2)$$

- Let's derive a distribution for the next sample x_{N+1}.

inference

- Clearly, the posterior for θ is now

$$p(\theta|D) \propto \underbrace{(\theta|\mu_0, \sigma_0^2)}_{\text{prior}} \cdot \underbrace{\prod_{n=1}^{N} (x_n|\theta, \sigma^2)}_{\text{likelihood}}$$

  which is a multiplication of $N+1$ Gaussians and is therefore also Gaussian distributed.

- Using the property that precisions and precision-weighted means add when Gaussians are multiplied, we can immediately write the posterior

$$p(\theta|D) = (\theta|\mu_N, \sigma_N^2)$$

  as

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \sum_n \frac{1}{\sigma^2}$$

$$\mu_N = \sigma_N^2 \left(\frac{1}{\sigma_0^2}\mu_0 + \sum_n \frac{1}{\sigma^2} x_n \right)$$

application: prediction of future sample

- We now have a posterior for the model parameters. Let's write down what we know about the next sample x_{N+1}.

$$\begin{aligned} p(x_{N+1}|D_N) &= \int p(x_{N+1}|\theta) p(\theta|D_N) \mathrm{d}\theta \\ &= \int (x_{N+1}|\theta, \sigma^2) \ (\theta|\mu_N, \sigma_N^2) \mathrm{d}\theta \\ &= (x_{N+1}|\mu_N, \sigma_N^2 + \sigma^2) \end{aligned}$$

- Uncertainty about x_{N+1} comprises both uncertainty about the parameter (σ_N^2) and observation noise σ^2.

Maximum Likelihood Estimation for the Gaussian

- In order to determine the maximum likelihood estimate of θ, we let $\sigma_0^2 \to \infty$ (leads to uniform prior for θ), yielding $\frac{1}{\sigma_N^2} = \frac{N}{\sigma^2}$ and consequently

$$\mu_{\mathrm{ML}} = \mu_N \Big|_{\sigma_0^2 \to \infty} = \sigma_N^2 \left(\frac{1}{\sigma^2} \sum_n x_n \right) = \frac{1}{N} \sum_{n=1}^{N} x_n$$

- As expected, having an expression for the maximum likelihood estimate, it is now possible to rewrite the (Bayesian) posterior mean for θ as (exercise)

$$\underset{\text{posterior}}{\mu_N} = \frac{\sigma^2}{N}\left(\frac{1}{\sigma_0^2}\mu_0 + \sum_n \frac{1}{\sigma^2}x_n\right)$$

$$= \underset{\text{prior}}{\mu_0} + \underset{\text{gain}}{\frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}} \cdot \underbrace{\left(\mu_{\mathrm{ML}} - \mu_0\right)}_{\text{prediction error}}$$

$$\text{correction}$$

- Hence, the posterior mean always lies somewhere between the prior mean μ_0 and the maximum likelihood estimate (the "data" mean) μ_{ML}.

Conditioning and Marginalization of a Gaussian

- Let $z = \begin{bmatrix} x \\ y \end{bmatrix}$ be jointly normal distributed as

$$p(z) = (z \mid \mu, \Sigma) = \left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix}\right)$$

- Since covariance matrices are by definition symmetric, it follows that Σ_x and Σ_y are symmetric and $\Sigma_{xy} = \Sigma_{yx}^T$.

- Let's factorize $p(z) = p(x,y)$ into $p(y \mid x) \cdot p(x)$ through conditioning and marginalization.

$$\text{conditioning: } p(y \mid x) = \left(y \mid \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x),\ \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}\right)$$

$$\text{marginalization: } p(x) = \left(x \mid \mu_x, \Sigma_x\right)$$

- proof: in Bishop pp.87-89

- Hence, conditioning and marginalization in Gaussians leads to Gaussians again. This is very useful for applications to Bayesian inference in jointly Gaussian systems.

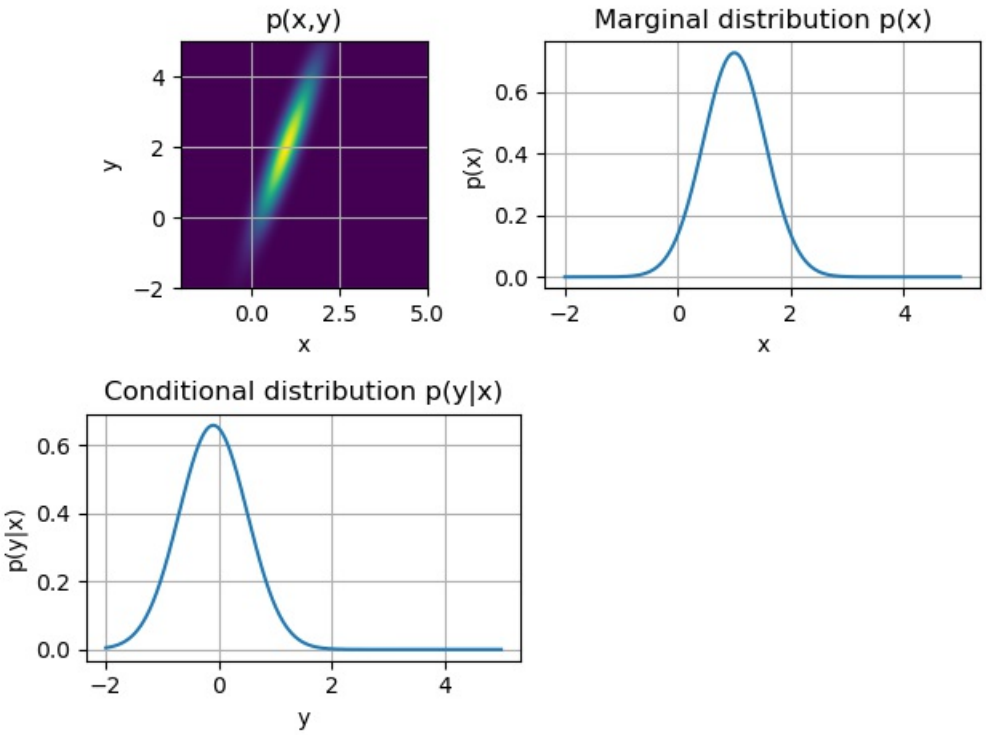
- With a natural parameterization of the Gaussian $p(z) = \quad _c(z \mid \eta, \Lambda)$ with precision matrix $\Lambda = \Sigma^{-1} = \begin{bmatrix} \Lambda_x & \Lambda_{xy} \\ \Lambda_{yx} & \Lambda_y \end{bmatrix}$, the conditioning operation results in a simpler result, see Bishop pg.90, eqs. 2.96 and 2.97.

- As an exercise, interpret the formula for the conditional mean ($\mathrm{E}[y \mid x] = \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x)$) as a prediction-correction operation.

CODE EXAMPLE

Plot of the joint, marginal, and conditional distributions.

```
using PyPlot, Distributions

μ = [1.0; 2.0]
Σ = [0.3 0.7;
     0.7 2.0]
joint = MvNormal(μ,Σ)
marginal_x = Normal(μ[1], sqrt(Σ[1,1]))

#Plot p(x,y)
subplot(221)
x_range = y_range = range(-2,stop=5,length=1000)
joint_pdf = [ pdf(joint, [x_range[i];y_range[j]]) for  j=1:length(y_range), i=1:length(x_range)]
imshow(joint_pdf, origin="lower", extent=[x_range[1], x_range[end], y_range[1], y_range[end]])
grid(); xlabel("x"); ylabel("y"); title("p(x,y)"); tight_layout()

# Plot p(x)
subplot(222)
plot(range(-2,stop=5,length=1000), pdf.(marginal_x, range(-2,stop=5,length=1000)))
grid(); xlabel("x"); ylabel("p(x)"); title("Marginal distribution p(x)"); tight_layout()

# Plot p(y|x)
x = 0.1
conditional_y_m = μ[2]+Σ[2,1]*inv(Σ[1,1])*(x-μ[1])
conditional_y_s2 = Σ[2,2] - Σ[2,1]*inv(Σ[1,1])*Σ[1,2]
conditional_y = Normal(conditional_y_m, sqrt.(conditional_y_s2))
subplot(223)
plot(range(-2,stop=5,length=1000), pdf.(conditional_y, range(-2,stop=5,length=1000)))
grid(); xlabel("y"); ylabel("p(y|x)"); title("Conditional distribution p(y|x)"); tight_layout()
```



As is clear from the plots, the conditional distribution is a renormalized slice from the joint distribution.

Example: Conditioning of Gaussian

- Consider (again) the system

$$p(x \mid \theta) = \quad (x \mid \theta, \sigma^2)$$

  with a Gaussian prior for θ:

$$p(\theta) = \quad (\theta \mid \mu_0, \sigma_0^2)$$

- Let $z = \begin{bmatrix} x \\ \theta \end{bmatrix}$. The distribution for z is then given by (Exercise)

$$p(z) = p\left(\begin{bmatrix} x \\ \theta \end{bmatrix}\right) = \quad \left(\begin{bmatrix} x \\ \theta \end{bmatrix} \middle| \begin{bmatrix} \mu_0 \\ \mu_0 \end{bmatrix}, \begin{bmatrix} \sigma_0^2 + \sigma^2 & \sigma_0^2 \\ \sigma_0^2 & \sigma_0^2 \end{bmatrix}\right)$$

- Direct substitution of the rule for Gaussian conditioning leads to the posterior (derivation as an Exercise):

$$p(\theta|x) = \quad \left(\theta \mid \mu_1, \sigma_1^2\right),$$

with

$$K = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} \qquad (K \text{ is called: Kalman gain})$$
$$\mu_1 = \mu_0 + K \cdot (x - \mu_0)$$
$$\sigma_1^2 = (1 - K)\sigma_0^2$$

- ⇒ Moral: For jointly Gaussian systems, we can do inference simply in one step by using the formulas for conditioning and marginalization.

Recursive Bayesian Estimation

- Consider the signal $x_t = \theta + \epsilon_t$, where $D_t = \left\{x_1, \ldots, x_t\right\}$ is observed sequentially (over time).
- Problem: Derive a recursive algorithm for $p(\theta|D_t)$, i.e., an update rule for (posterior) $p(\theta|D_t)$ based on (prior) $p(\theta|D_{t-1})$ and (new observation) x_t.

Model specification

- Let's define the estimate after t observations (i.e., our solution) as $p(\theta|D_t) = \quad (\theta \mid \mu_t, \sigma_t^2)$.
- We define the joint distribution for θ and x_t, given background D_{t-1}, by

$$p(x_t, \theta \mid D_{t-1}) = p(x_t|\theta)\, p(\theta|D_{t-1})$$
$$= \quad (x_t \mid \theta, \sigma^2) \quad (\theta \mid \mu_{t-1}, \sigma_{t-1}^2)$$

likelihood prior

Inference

- Use Bayes rule,

$$
\begin{aligned}
p(\theta|D_t) &= p(\theta|x_t, D_{t-1}) \\
&\propto p(x_t, \theta|D_{t-1}) \\
&= p(x_t|\theta)\, p(\theta|D_{t-1}) \\
&= (x_t|\theta, \sigma^2) \quad (\theta|\mu_{t-1}, \sigma^2_{t-1}) \\
&= (\theta|x_t, \sigma^2) \quad (\theta|\mu_{t-1}, \sigma^2_{t-1}) \text{ (note this trick)} \\
&= (\theta|\mu_t, \sigma^2_t) \text{ (use Gaussian multiplication formula SRG-6)}
\end{aligned}
$$

  with

$$
\begin{aligned}
K_t &= \frac{\sigma^2_{t-1}}{\sigma^2_{t-1} + \sigma^2} \quad \text{(Kalman gain)} \\
\mu_t &= \mu_{t-1} + K_t \cdot (x_t - \mu_{t-1}) \\
\sigma^2_t &= \left(1 - K_t\right)\sigma^2_{t-1}
\end{aligned}
$$

- This linear sequential estimator of mean and variance in Gaussian observations is called a Kalman Filter.

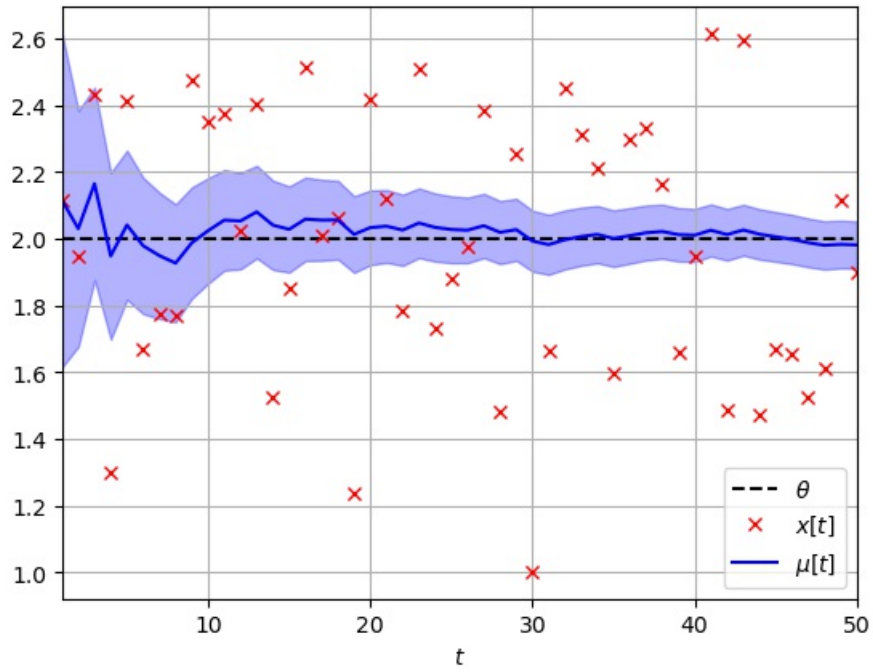
- Note that the uncertainty about θ decreases over time (since $0 < (1 - K_t) < 1$). Since we assume that the statistics of the system do not change (stationarity), each new sample provides new information.

- Recursive Bayesian estimation is the basis for adaptive signal processing algorithms such as Least Mean Squares (LMS) and Recursive Least Squares (RLS).

CODE EXAMPLE

Let's implement the Kalman filter described above. We'll use it to recursively estimate the value of θ based on noisy observations.

```
using PyPlot

N = 50                          # Number of observations
θ = 2.0                         # True value of the variable we want to estimate
σ_ϵ2 = 0.25                     # Observation noise variance
x = sqrt(σ_ϵ2) * randn(N) .+ θ # Generate N noisy observations of θ

t = 0
μ = fill!(Vector{Float64}(undef,N), NaN)     # Means of p(θ|D) over time
σ_μ2 = fill!(Vector{Float64}(undef,N), NaN) # Variances of p(θ|D) over time

function performKalmanStep()
    # Perform a Kalman filter step, update t, μ, σ_μ2
    global t += 1
    if t>1 # Use posterior from prev. step as prior
        K = σ_μ2[t-1] / (σ_ϵ2 + σ_μ2[t-1]) # Kalman gain
        μ[t] = μ[t-1] + K*(x[t] - μ[t-1])  # Update mean using (1)
        σ_μ2[t] = σ_μ2[t-1] * (1.0-K)      # Update variance using (2)
    elseif t==1 # Use prior
        # Prior p(θ) = N(0,1000)
        K = 1000.0 / (σ_ϵ2 + 1000.0) # Kalman gain
        μ[t] = 0 + K*(x[t] - 0)      # Update mean using (1)
        σ_μ2[t] = 1000 * (1.0-K)     # Update variance using (2)
    end
end

while t<N
    performKalmanStep()
end

# Plot the 'true' value of θ, noisy observations x, and the recursively updated posterior p(θ|D)
t = collect(1:N)
plot(t, θ*ones(N), "k--")
plot(t, x, "rx")
plot(t, μ, "b-")
fill_between(t, μ-sqrt.(σ_μ2), μ+sqrt.(σ_μ2), color="b", alpha=0.3)
legend([L"\theta", L"x[t]", L"\mu[t]"])
xlim((1, N)); xlabel(L"t"); grid()
```



The shaded area represents 2 standard deviations of posterior $p(\theta|D)$. The variance of the posterior is guaranteed to decrease monotonically for the standard Kalman filter.

Product of Normally Distributed Variables

- (We've seen that) the sum of two Gausssian distributed variables is also Gaussian distributed.

- Has the product of two Gaussian distributed variables also a Gaussian distribution?

- No! In general this is a difficult computation. As an example, let's compute $p(z)$ for $Z = XY$ for the special case that $X \sim (0,1)$ and $Y \sim (0,1)$.

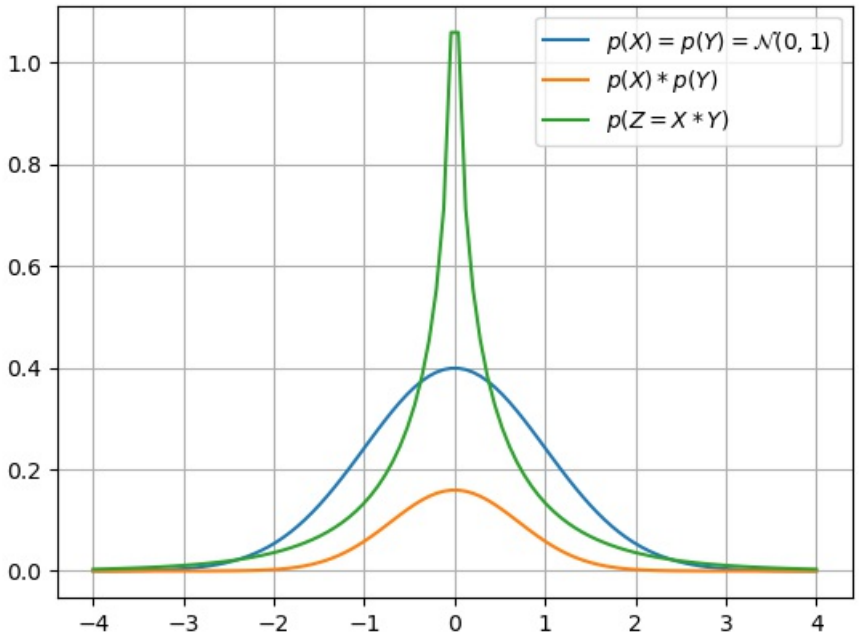
$$\begin{aligned} p(z) &= \int_{X,Y} p(z \mid x,y)\, p(x,y)\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{2\pi}\int \delta(z-xy)\, e^{-(x^2+y^2)/2}\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{\pi}\int_0^\infty \frac{1}{x} e^{-(x^2+z^2/x^2)/2}\, \mathrm{d}x \\ &= \frac{1}{\pi}\mathrm{K}_0(|z|). \end{aligned}$$

  where $\mathrm{K}_n(z)$ is a modified Bessel function of the second kind.

CODE EXAMPLE

- We plot $p(Z = XY)$ and $p(X)p(Y)$ for $X \sim (0,1)$ and $Y \sim (0,1)$ to give an idea of how these distributions differ.

```
using PyPlot, Distributions, SpecialFunctions
X = Normal(0,1)
Y = Normal(0,1)
pdf_product_std_normals(z::Vector) = (besselk.(0, abs.(z))./π)
range1 = collect(range(-4,stop=4,length=100))
plot(range1, pdf.(X, range1))
plot(range1, pdf.(X,range1).*pdf.(Y,range1))
plot(range1, pdf_product_std_normals(range1))
legend([L"p(X)=p(Y)=\mathcal{N}(0,1)", L"p(X)*p(Y)",L"p(Z=X*Y)"]); grid()
```

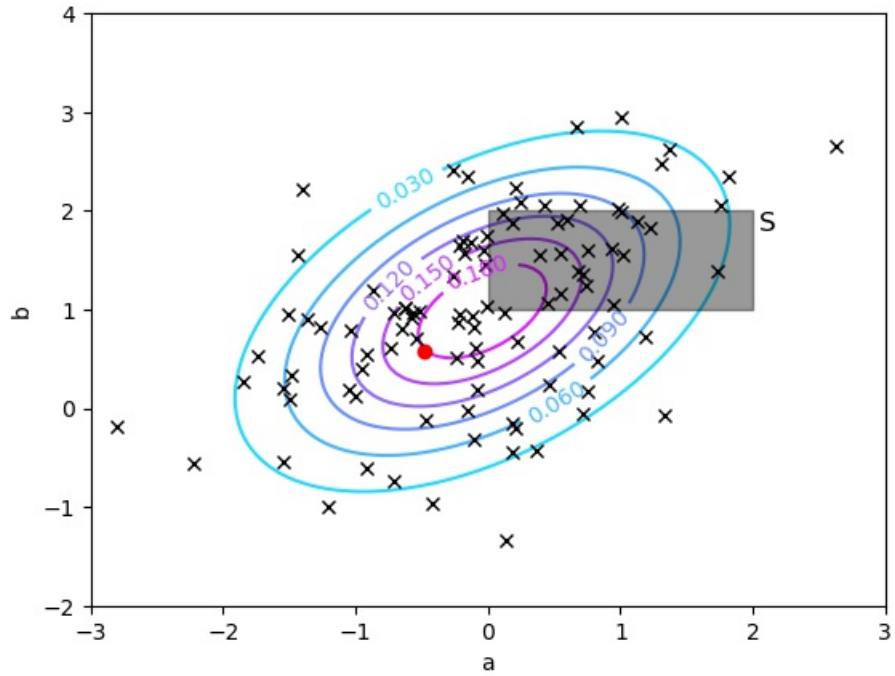


- In short, Gaussian-distributed variables remain Gaussian in linear systems, but this is not the case in non-linear systems.

Solution to Example Problem

We apply maximum likelihood estimation to fit a 2-dimensional Gaussian model (m) to data set D. Next, we evaluate $p(x. \in S|m)$ by (numerical) integration of the Gaussian pdf over S: $p(x. \in S|m) = \int_S p(x|m)\mathrm{d}x.$

```
using HCubature, LinearAlgebra# Numerical integration package
# Maximum likelihood estimation of 2D Gaussian
N = length(sum(D,dims=1))
μ = 1/N * sum(D,dims=2)[:,1]
D_min_μ = D - repeat(μ, 1, N)
Σ = Hermitian(1/N * D_min_μ*D_min_μ')
m = MvNormal(μ, convert(Matrix, Σ));

# # Contour plot of estimated Gaussian density
A = Matrix{Float64}(undef,100,100); B = Matrix{Float64}(undef,100,100)
density = Matrix{Float64}(undef,100,100)
for i=1:100
    for j=1:100
        A[i,j] = a = (i-1)*6/100 .- 2
        B[i,j] = b = (j-1)*6/100 .- 3
        density[i,j] = pdf(m, [a,b])
    end
end
c = contour(A, B, density, 6, zorder=1)
PyPlot.set_cmap("cool")
clabel(c, inline=1, fontsize=10)

# Plot observations, x·, and the countours of the estimated Gausian density
plotObservations(D)
plot(x_dot[1], x_dot[2], "ro")

# Numerical integration of p(x|m) over S:
(val,err) = hcubature((x)->pdf(m,x), [0., 1.], [2., 2.])
println("p(x·∈S|m) ≈ $(val)")
```




```
p(x·∈S|m) ≈ 0.2032961495212706
```

Summary

- A linear transformation of a Gaussian-distributed (potentially multivariate) variable remains Gaussian.
- Bayesian inference with Gaussian prior and likelihood leads to an analytically computable Gaussian posterior.
- Here's a nice summary of Gaussian calculations.

OPTIONAL SLIDES

Inference for the Precision Parameter of the Gaussian

- Again, we consider an observed data set $D = \{x_1, x_2, \ldots, x_N\}$ and try to explain these data by a Gaussian distribution.

- We discussed earlier Bayesian inference for the mean with a given variance. Now we will derive a posterior for the variance if the mean is given. (Technically, we will do the derivation for a precision parameter $\lambda = \sigma^{-2}$, since the discussion is a bit more straightforward for the precision parameter).

model specification

- The likelihood for the precision parameter is

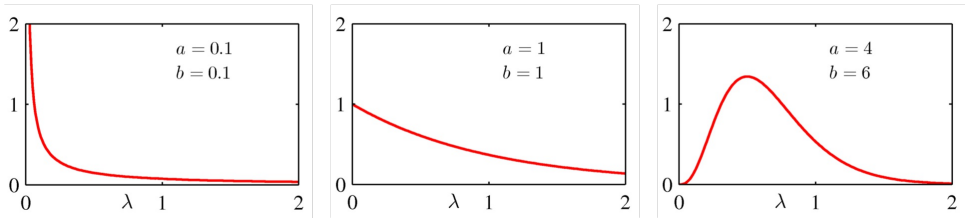
$$p(D \mid \lambda) = \prod_{n=1}^{N} \left(x_n \mid \mu, \lambda^{-1}\right)$$

$$\propto \lambda^{N/2} \exp\left\{-\frac{\lambda}{2} \sum_{n=1}^{N} \left(x_n - \mu\right)^2\right\}$$

- The conjugate distribution for this function of λ is the Gamma distribution, given by

$$p(\lambda \mid a, b) = \mathrm{Gam}(\lambda \mid a, b) \triangleq \frac{1}{\Gamma(a)} b^a \lambda^{a-1} \exp\{-b\lambda\},$$

  where $a > 0$ and $b > 0$ are known as the shape and rate parameters, respectively.



- (Bishop fig.2.13). Plots of the Gamma distribution $\mathrm{Gam}(\lambda \mid a, b)$ for different values of a and b.

- The mean and variance of the Gamma distribution evaluate to $\mathrm{E}(\lambda) = \frac{a}{b}$ and $\mathrm{var}[\lambda] = \frac{a}{b^2}$.

inference

- We will consider a prior $p(\lambda) = \mathrm{Gam}\left(\lambda \mid a_0, b_0\right)$, which leads by Bayes rule to the posterior

$$p(\lambda \mid D) \propto \underbrace{\lambda^{N/2} \exp\left\{-\frac{\lambda}{2}\sum_{n=1}^{N}\left(x_n - \mu\right)^2\right\}}_{\text{likelihood}} \cdot \underbrace{\frac{1}{\Gamma(a_0)} b_0^{a_0} \lambda^{a_0-1} \exp\left\{-b_0\lambda\right\}}_{\text{prior}}$$

$$\propto \mathrm{Gam}\left(\lambda \mid a_N, b_N\right)$$

  with

$$a_N = a_0 + \frac{N}{2}$$

$$b_N = b_0 + \frac{1}{2}\sum_n \left(x_n - \mu\right)^2$$

- Hence the posterior is again a Gamma distribution. By inspection of B-2.150 and B-2.151, we deduce that we can interpret $2a_0$ as the number of a priori (pseudo-)observations.

- Since the most uninformative prior is given by $a_0 = b_0 \to 0$, we can derive the maximum likelihood estimate for the precision as

$$\lambda_{\mathrm{ML}} = \mathrm{E}[\lambda]_{a_0 = b_0 \to 0} = \left.\frac{a_N}{b_N}\right|_{a_0 = b_0 \to 0} = \frac{N}{\sum_{n=1}^{N}\left(x_n - \mu\right)^2}$$

- In short, if we do density estimation with a Gaussian distribution $\left(x_n \mid \mu, \sigma^2\right)$ for an observed data set $D = \{x_1, x_2, \ldots, x_N\}$, the maximum likelihood estimates for μ and σ^2 are given by

$$\mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

$$\sigma^2_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} \left(x_n - \mu_{\mathrm{ML}}\right)^2$$

- These estimates are also known as the sample mean and sample variance respectively.

Discrete Data and the Multinomial Distribution

Preliminaries

- Goal
  - Simple Bayesian and maximum likelihood-based density estimation for discretely valued data sets
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 67-70, 74-76, 93-94

Discrete Data: the 1-of-K Coding Scheme

- Consider a coin-tossing experiment with outcomes $x \in \{0, 1\}$ (tail and head) and let $0 \leq \mu \leq 1$ represent the probability of heads. This model can written as a Bernoulli distribution:

$$p(x \mid \mu) = \mu^x (1 - \mu)^{1-x}$$

  - Note that the variable x acts as a (binary) selector for the tail or head probabilities. Think of this as an 'if'-statement in programming.

- Now consider a K-sided coin (e.g., a six-faced die (pl.: dice)). How should we encode outcomes?

- Option 1: $x \in \{1, 2, \ldots, K\}$.
  - E.g., for $K = 6$, if the die lands on the 3rd face $\Rightarrow x = 3$.
- Option 2: $x = (x_1, \ldots, x_K)^T$ with binary selection variables

$$x_k = \begin{cases} 1 & \text{if die landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

  - E.g., for $K = 6$, if the die lands on the 3rd face $\Rightarrow x = (0, 0, 1, 0, 0, 0)^T$.
  - This coding scheme is called a 1-of-K or one-hot coding scheme.

- It turns out that the one-hot coding scheme is mathematically more convenient!

- Consider a K-sided die. We use a one-hot coding scheme. Assume the probabilities $p(x_k = 1) = \mu_k$ with $\sum_k \mu_k = 1$. The data generating distribution is then (note the similarity to the Bernoulli distribution)

$$p(x \mid \mu) = \mu_1^{x_1} \mu_2^{x_2} \cdots \mu_K^{x_K} = \prod_{k=1}^{K} \mu_k^{x_k}$$

- This generalized Bernoulli distribution is called the categorical distribution (or sometimes the 'multi-noulli' distribution).

Bayesian Density Estimation for a Loaded Die

- Now let's proceed with Bayesian density estimation $p(x \mid \theta)$ for an observed data set $D = \{x_1, \ldots, x_N\}$ of N IID rolls of a K-sided die, with

$$x_{nk} = \begin{cases} 1 & \text{if the } n\text{th throw landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

Model specification

- The data generating PDF is

$$p(D \mid \mu) = \prod_n \prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}} = \prod_k \mu_k^{m_k}$$

  where $m_k = \sum_n x_{nk}$ is the total number of occurrences that we 'threw' k eyes. Note that $\sum_k m_k = N$.

  - This distribution depends on the observations only through the quantities $\{m_k\}$.

- We need a prior for the parameters $\mu = (\mu_1, \mu_2, \ldots, \mu_K)$. In the binary coin toss example, we used a beta distribution that was conjugate with the binomial and forced us to choose prior pseudo-counts.
- The generalization of the beta prior to the K parameters $\{\mu_k\}$ is the Dirichlet distribution:

$$p(\mu \mid \alpha) = \mathrm{Dir}(\mu \mid \alpha) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_K)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1}$$

  - Again, you can interpret α_k as the prior number of (pseudo-)observations that the die landed on the kth face.

Inference for $\{\mu_k\}$

- The posterior for $\{\mu_k\}$ can be obtained through Bayes rule:

$$\begin{aligned} p(\mu \mid D, \alpha) &\propto p(D \mid \mu) \cdot p(\mu \mid \alpha) \\ &\propto \prod_k \mu_k^{m_k} \cdot \prod_k \mu_k^{\alpha_k - 1} \\ &= \prod_k \mu_k^{\alpha_k + m_k - 1} \\ &\propto \mathrm{Dir}(\mu \mid \alpha + m) \\ &= \frac{\Gamma\left(\sum_k (\alpha_k + m_k)\right)}{\Gamma(\alpha_1 + m_1)\Gamma(\alpha_2 + m_2)\cdots\Gamma(\alpha_K + m_K)} \prod_{k=1}^{K} \mu_k^{\alpha_k + m_k - 1} \end{aligned}$$

where $m = (m_1, m_2, \ldots, m_K)^T$.

- Again, we recognize the α_k's as prior pseudo-counts and the Dirichlet distribution shows to be a conjugate prior to the categorical/multinomial:

$$\underset{\text{posterior}}{\text{Dirichlet}} \propto \underset{\text{likelihood}}{\text{categorical}} \cdot \underset{\text{prior}}{\text{Dirichlet}}$$

- This is actually a generalization of the conjugate relation that we found for the binary coin toss:

$$\underset{\text{posterior}}{\text{Beta}} \propto \underset{\text{likelihood}}{\text{binomial}} \cdot \underset{\text{prior}}{\text{Beta}}$$

Prediction of next toss for the loaded die

- Let's apply what we have learned about the loaded die to compute the probability that we throw the kth face at the next toss.

$$\begin{aligned} p(x_{\cdot,k} = 1 \mid D) &= \int p(x_{\cdot,k} = 1 \mid \mu)\, p(\mu \mid D)\, \mathrm{d}\mu \\ &= \int_0^1 \mu_k \times \quad (\mu \mid \alpha + m)\, \mathrm{d}\mu \\ &= \mathrm{E}\left[\mu_k\right] \\ &= \frac{m_k + \alpha_k}{N + \sum_k \alpha_k} \end{aligned}$$

- (You can find the mean of the Dirichlet distribution at its Wikipedia site).
- This result is simply a generalization of Laplace's rule of succession.

Categorical, Multinomial and Related Distributions

- In the above derivation, we noticed that the data generating distribution for N die tosses $D = \{x_1, \ldots, x_N\}$ only depends on the data frequencies m_k:

$$p(D|\mu) = \prod_n \prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}} = \prod_k \mu_k^{m_k}$$

categorical dist.

- A related distribution is the distribution over data frequency observations $D_m = \{m_1, \ldots, m_K\}$, which is called the multinomial distribution,

$$p(D_m|\mu) = \frac{N!}{m_1!m_2!\ldots m_K!} \prod_k \mu_k^{m_k}.$$

- Verify for yourself that (Exercise):
  - the categorial distribution is a special case of the multinomial for $N = 1$.
  - the Bernoulli is a special case of the categorial distribution for $K = 2$.
  - the binomial is a special case of the multinomial for $K = 2$.

Maximum Likelihood Estimation for the Multinomial

Maximum likelihood as a special case of Bayesian estimation

- We can get the maximum likelihood estimate $\hat{\mu}_k$ for μ_k based on N throws of a K-sided die from the Bayesian framework by using a uniform prior for μ and taking the mode of the posterior for μ:

$$\begin{aligned}
\hat{\mu}_k &= \arg\max_{\mu_k} p(D|\mu) \\
&= \arg\max_{\mu_k} p(D|\mu) \cdot \text{Uniform}(\mu) \\
&= \arg\max_{\mu_k} p(D|\mu) \cdot \text{Dir}(\mu|\alpha)|_{\alpha=(1,1,\ldots,1)} \\
&= \arg\max_{\mu_k} p(\mu|D,\alpha)|_{\alpha=(1,1,\ldots,1)} \\
&= \arg\max_{\mu_k} \text{Dir}(\mu|m+\alpha)|_{\alpha=(1,1,\ldots,1)} \\
&= \frac{m_k}{\sum_k m_k} = \frac{m_k}{N}
\end{aligned}$$

where we used the fact that the maximum of the Dirichlet distribution $\text{Dir}(\{\alpha_1, \ldots, \alpha_K\})$ is obtained at $(\alpha_k - 1)/(\sum_k \alpha_k - K)$.

Maximum likelihood estimation by optimizing a constrained log-likelihood

- Of course, we shouldn't have to go through the full Bayesian framework to get the maximum likelihood estimate. Alternatively, we can find the maximum of the likelihood directly.
- The log-likelihood for the multinomial distribution is given by

$$\mathrm{L}(\mu) \triangleq \log p(D_m \,|\, \mu) \propto \log \prod_k \mu_k^{m_k} = \sum_k m_k \log \mu_k$$

- When doing ML estimation, we must obey the constraint $\sum_k \mu_k = 1$, which can be accomplished by a Lagrange multiplier (see Bishop App.E). The augmented log-likelihood with Lagrange multiplier is then

$$\mathrm{L}'(\mu) = \sum_k m_k \log \mu_k + \lambda \cdot \left(1 - \sum_k \mu_k\right)$$

- Set derivative to zero yields the sample proportion for μ_k

$$\nabla_{\mu_k} \mathrm{L}' = \frac{m_k}{\hat{\mu}_k} - \lambda \overset{!}{=} 0 \;\Rightarrow\; \hat{\mu}_k = \frac{m_k}{N}$$

  where we get λ from the constraint

$$\sum_k \hat{\mu}_k = \sum_k \frac{m_k}{\lambda} = \frac{N}{\lambda} \overset{!}{=} 1$$

Recap Maximum Likelihood Estimation for Gaussian and Multinomial Distributions

Given N IID observations $D = \{x_1, \ldots, x_N\}$.

- For a multivariate Gaussian model $p(x_n \,|\, \theta) = \quad (x_n \,|\, \mu, \Sigma)$, we obtain ML estimates

$$\hat{\mu} = \frac{1}{N} \sum_n x_n$$

$$\hat{\Sigma} = \frac{1}{N} \sum_n (x_n - \hat{\mu})(x_n - \hat{\mu})^T$$

- For discrete outcomes modeled by a 1-of-K categorical distribution we find

$$\hat{\mu}_k = \frac{1}{N} \sum_n x_{nk} \quad \left(= \frac{m_k}{N} \right)$$

- Note the similarity for the means between discrete and continuous data.

OPTIONAL SLIDES

Some Useful Matrix Calculus

When doing derivatives with matrices, e.g. for maximum likelihood estimation, it will be helpful to be familiar with some matrix calculus. We shortly recapitulate used formulas here.

- We define the gradient of a scalar function $f(A)$ w.r.t. an $n \times k$ matrix A as

$$\nabla_A f \triangleq \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \cdots & \frac{\partial f}{\partial a_{1k}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \cdots & \frac{\partial f}{\partial a_{2k}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f}{\partial a_{n1}} & \frac{\partial f}{\partial a_{n2}} & \cdots & \frac{\partial f}{\partial a_{nk}} \end{bmatrix}$$

- The following formulas are useful (see Bishop App.-C)

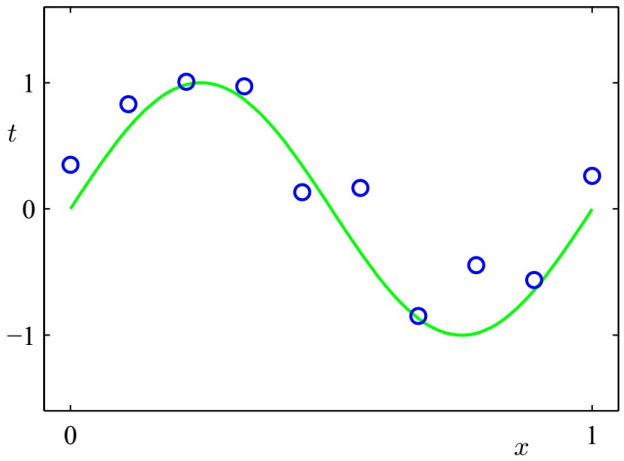
$$\begin{aligned} |A^{-1}| &= |A|^{-1} \\ \nabla_A \log|A| &= (A^T)^{-1} = (A^{-1})^T \\ \mathrm{Tr}[ABC] &= \mathrm{Tr}[CAB] = \mathrm{Tr}[BCA] \\ \nabla_A \mathrm{Tr}[AB] &= \nabla_A \mathrm{Tr}[BA] = B^T \\ \nabla_A \mathrm{Tr}[ABA^T] &= A(B + B^T) \\ \nabla_x x^T Ax &= (A + A^T)x \\ \nabla_X a^T Xb &= \nabla_X \mathrm{Tr}[ba^T X] = ab^T \end{aligned}$$

Regression

Preliminaries

- Goal
  - Introduction to Bayesian (Linear) Regression
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 152-158

Regression - Illustration



Given a set of (noisy) data measurements, find the 'best' relation between an input variable $x \in \mathrm{R}^D$ and input-dependent outcomes $y \in \mathrm{R}$

Regression vs Density Estimation

- Observe N IID data pairs $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathrm{R}^M$ and $y_n \in \mathrm{R}$.
- Assume that we are interested in (a model for) the responses y_n for given inputs x_n?, I.o.w. we are interested in building a model for the conditional distribution $p(y|x)$.
- Note that, since $p(x,y) = p(y|x)\,p(x)$, building a model $p(y|x)$ is similar to density estimation with the assumption that x is drawn from a uniform distribution.

Bayesian Linear Regression

- Next, we discuss (1) model specification, (2) Inference and (3) a prediction application for a Bayesian linear regression problem.

1. Model Specification

- In a regression model, we try to 'explain the data' by a purely deterministic term $f(x_n, w)$, plus a purely random term ϵ_n for 'unexplained noise',

$$y_n = f(x_n, w) + \epsilon_n$$

- In a linear regression model, i.e., linear w.r.t. the parameters w, we assume that

$$f(x_n, w) = \sum_{j=0}^{M-1} w_j \phi_j(x_n) = w^T \phi(x_n)$$

  where $\phi_j(x)$ are called basis functions.

  - For notational simplicity, from now on we will assume $f(x_n, w) = w^T x_n$, with $x_n \in \mathrm{R}^M$.

- In ordinary linear regression , the noise process ϵ_n is zero-mean Gaussian with constant variance, i.e.

$$y_n = w^T x_n + \quad (0, \beta^{-1}) .$$

- Hence, given a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the likelihood for an ordinary linear regression model is

$$\begin{aligned} p(y \mid \mathbf{X}, w, \beta) = & \quad (y \mid \mathbf{X}w, \beta^{-1}\mathbf{I}) \\ = & \prod_n \quad (y_n \mid w^T x_n, \beta^{-1}) \end{aligned}$$

where $w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{pmatrix}$, the $(N \times M)$-dim matrix $\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} = \begin{pmatrix} x_{11}, x_{12}, \ldots, x_{1M} \\ x_{21}, x_{22}, \ldots, x_{2M} \\ \vdots \\ x_{N1}, x_{N2}, \ldots, x_{NM} \end{pmatrix}$ and $y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$.

- For full Bayesian learning we should also choose a prior $p(w)$:

$$p(w \mid \alpha) = \quad (w \mid 0, \alpha^{-1}\mathbf{I})$$

- For simplicity, we will assume that α and β are known.

2. Inference

- We'll do Bayesian inference for the parameters w.

$$\begin{aligned} p(w|D) &\propto p(D|w) \cdot p(w) \\ &= \mathcal{N}(y \mid \mathbf{X}w, \beta^{-1}\mathbf{I}) \cdot \mathcal{N}(w \mid 0, \alpha^{-1}\mathbf{I}) \\ &\propto \exp\Big(-\frac{\beta}{2}(y-\mathbf{X}w)^T(y-\mathbf{X}w) - \frac{\alpha}{2}w^Tw\Big) \\ &= \exp\Big(-\frac{1}{2}w^T\underbrace{(\beta\mathbf{X}^T\mathbf{X}+\alpha\mathbf{I})}_{\Lambda_N}w + \underbrace{(\beta\mathbf{X}^Ty)}_{\eta_N}{}^Tw - \frac{\beta}{2}y^Ty\Big) \\ &\propto \mathcal{N}_c\big(w \mid \eta_N, \Lambda_N\big) \end{aligned}$$

with natural parameters (see the [natural parameterization of Gaussian]):

$$\begin{aligned} \eta_N &= \beta\mathbf{X}^Ty \\ \Lambda_N &= \beta\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I} \end{aligned}$$

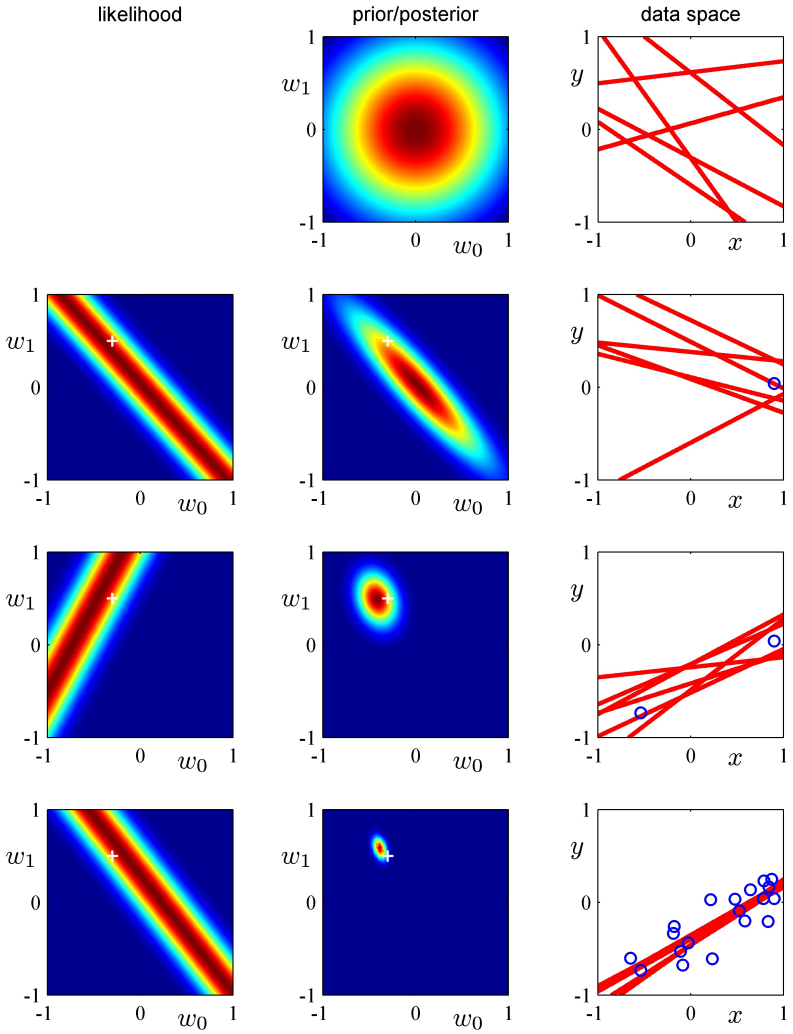
- Or equivalently (in the moment parameterization of the Gaussian):

$$\begin{aligned} p(w|D) &= \mathcal{N}\big(w \mid m_N, S_N\big) \\ m_N &= \beta S_N\mathbf{X}^Ty \\ S_N &= \big(\alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X}\big)^{-1} \end{aligned}$$

- Note that B-3.53 and B-3.54 combine to

$$m_N = \Big(\frac{\alpha}{\beta}\mathbf{I} + \mathbf{X}^T\mathbf{X}\Big)^{-1}\mathbf{X}^Ty\,.$$

- (Bishop Fig.3.7) Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, w) = w_0 + w_1 x$. (Bishop Fig.3.7, detailed description at Bishop, pg.154.)



3. Application: predictive distribution

- Assume we are interested in the distribution $p(y_\cdot \mid x_\cdot, D)$ for a new input $x_\cdot$. This can be worked out as (exercise B-3.10)

$$
\begin{aligned}
p(y_\cdot \mid x_\cdot, D) &= \int p(y_\cdot \mid x_\cdot, w) p(w \mid D)\, \mathrm{d}w \\
&= \int \; (y_\cdot \mid w^T x_\cdot, \beta^{-1}) \;\; (w \mid m_N, S_N)\, \mathrm{d}w \\
&= \int \; (y_\cdot \mid z, \beta^{-1}) \;\; (z \mid x_\cdot^T m_N, x_\cdot^T S_N x_\cdot)\, \mathrm{d}z \quad (\text{sub. } z = x_\cdot^T w) \\
&= \int \; (z \mid y_\cdot, \beta^{-1}) \;\; (z \mid x_\cdot^T m_N, x_\cdot^T S_N x_\cdot)\, \mathrm{d}z \\
&\qquad \text{Use Gaussian product formula SRG-6} \\
&= \int \; \left(y_\cdot \mid m_N^T x_\cdot, \sigma_N^2(x_\cdot)\right) \;\; \left(z \mid m_z, S_z\right) \mathrm{d}z \\
&\qquad\qquad\qquad\qquad \text{integrate this out} \\
&= \; \left(y_\cdot \mid m_N^T x_\cdot, \sigma_N^2(x_\cdot)\right)
\end{aligned}
$$

with

$$\sigma_N^2(x_\cdot) = \beta^{-1} + x_\cdot^T S_N x_\cdot$$

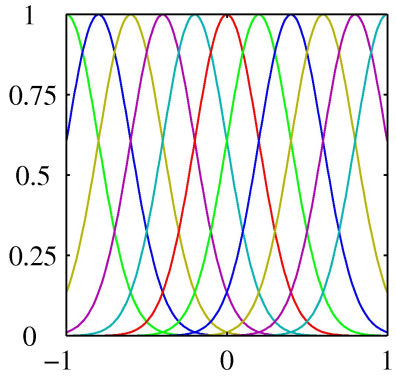
- So, the uncertainty $\sigma_N^2(x_\cdot)$ about the output contains comprises both uncertainty about the process (β^{-1}) and about the model parameters ($x_\cdot^T S_N x_\cdot$).

Example Predictive Distribution

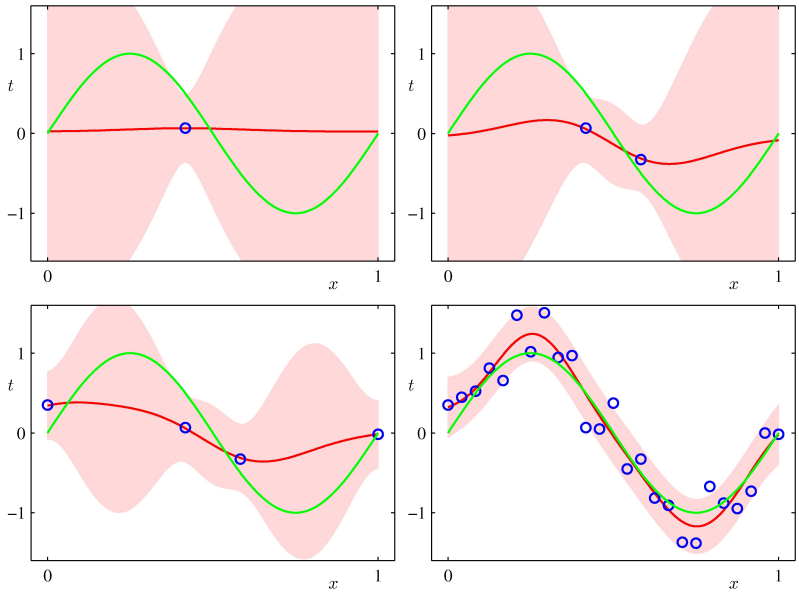
- As an example, let's do Bayesian Linear Regression for a synthetic sinusoidal data set and a model with 9 Gaussian basis functions

$$y = \sum_{m=1}^{9} w_m \phi_m(x_n) + \quad (0, \beta^{-1})$$

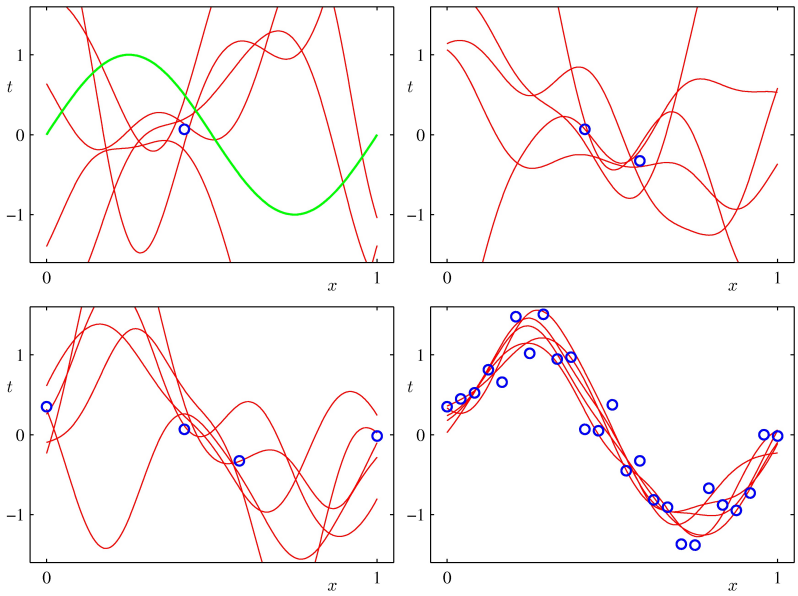
$$\phi_m(x_n) = \exp\left(\frac{x_n - \mu_m}{\sigma^2}\right)$$



- The predictive distributions for y are shown in the following plots (Bishop, Fig.3.8)



- And some plots of draws of posteriors for the functions $w^T\phi(x)$ (Bishop, Fig.3.9)



Maximum Likelihood Estimation for Linear Regression Model

- Recall the posterior mean for the weight vector

$$m_N = \left(\frac{\alpha}{\beta}\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T y$$

  where α is the prior precision for the weights.

- The Maximum Likelihood solution for w is obtained by letting $\alpha \to 0$, which leads to

$$\hat{w}_{\mathrm{ML}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

- The matrix $\mathbf{X}^\dagger \equiv (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is also known as the Moore–Penrose pseudo-inverse (which is sort-of-an-inverse for non-square matrices).

- Note that if we have fewer training samples than input dimensions, i.e., if $N < M$, then $\mathbf{X}^T\mathbf{X}$ will not be invertible and maximum likelihood blows up. The Bayesian solution does not suffer from this problem.

Least-Squares Regression

- (You may say that) we don't need to work with probabilistic models. E.g., there's also the deterministic least-squares solution: minimize sum of squared errors,

$$\hat{w}_{\mathrm{LS}} = \arg\min_w \sum_n \left(y_n - w^T x_n\right)^2 = \arg\min_w (y - \mathbf{X}w)^T(y - \mathbf{X}w)$$

- Setting the gradient $\frac{\partial (y-\mathbf{X}w)^T(y-\mathbf{X}w)}{\partial w} = -2\mathbf{X}^T(y-\mathbf{X}w)$ to zero yields the so-called normal equations $\mathbf{X}^T\mathbf{X}\hat{w}_{\text{LS}} = \mathbf{X}^T y$ and consequently

$$\hat{w}_{\text{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

which is the same answer as we got for the maximum likelihood weights $\hat{w}_{\text{ML}}$.

- $\Rightarrow$ Least-squares regression ($\hat{w}_{\text{LS}}$) corresponds to the (probabilistic) maximum likelihood solution ($\hat{w}_{\text{ML}}$) if the probabilistic model includes the following assumptions:

  1. The observations are independently and identically distributed (IID) (this determines how errors are combined), and
  2. The noise signal $\epsilon_n \sim \quad (0, \beta^{-1})$ is Gaussian distributed (determines the error metric)

- If you use the Least-Squares method, you cannot see (nor modify) these assumptions. The probabilistic method forces you to state all assumptions explicitly!

Not Identically Distributed Data

- Let's do an example regarding changing our assumptions. What if we assume that the variance of the measurement error varies with the sampling index, $\epsilon_n \sim \quad (0, \beta_n^{-1})$?

- The likelihood is now (using $\Lambda \triangleq \text{diag}(\beta_n)$)

$$p(y \mid \mathbf{X}, w, \Lambda) = \quad (y \mid \mathbf{X}w, \Lambda^{-1}).$$

- Combining this likelihood with the prior $p(w) = \quad (w \mid 0, \alpha^{-1}\mathbf{I})$ leads to a posterior

$$\begin{aligned} p(w \mid D) &\propto p(D \mid w) \cdot p(w) \\ &= \quad (y \mid \mathbf{X}w, \Lambda^{-1}\mathbf{I}) \cdot \quad (w \mid 0, \alpha^{-1}\mathbf{I}) \\ &\propto \exp\left\{\frac{1}{2}(y-\mathbf{X}w)^T\Lambda(y-\mathbf{X}w) + \frac{\alpha}{2}w^T w\right\} \\ &\propto \quad \left(w \mid m_N, S_N\right) \end{aligned}$$

with

$$\begin{aligned} m_N &= S_N\mathbf{X}^T\Lambda y \\ S_N &= \left(\alpha\mathbf{I} + \mathbf{X}^T\Lambda\mathbf{X}\right)^{-1} \end{aligned}$$

- And maximum likelihood solution

$$\hat{w}_{\text{ML}} = m_N\big|_{\alpha\to 0} = \left(\mathbf{X}^T\Lambda\mathbf{X}\right)^{-1}\mathbf{X}^T\Lambda y$$

- This maximum likelihood solution is also called the Weighted Least Squares (WLS) solution. (Note that we just stumbled upon it, the crucial aspect is appropriate model specification!)

- Note also that the dimension of Λ grows with the number of data points. In general, models for which the number of parameters grow as the number of observations increase are called non-parametric models.

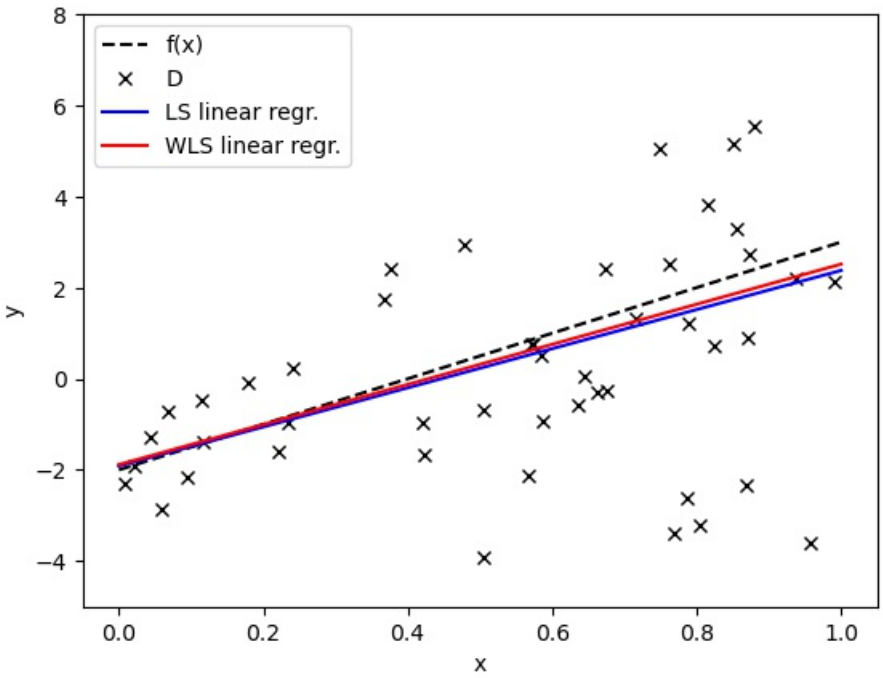
CODE EXAMPLE

We'll compare the Least Squares and Weighted Least Squares solutions for a simple linear regression model with input-dependent noise:

$$
\begin{aligned}
x &\sim \text{Unif}[0,1] \\
y \mid x &\sim \mathcal{N}(f(x), v(x)) \\
f(x) &= 5x - 2 \\
v(x) &= 10e^{2x^2} - 9.5 \\
&= \{(x_1, y_1), \ldots, (x_N, y_N)\}
\end{aligned}
$$

```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate();
IJulia.clear_output();
```

```
using PyPlot, LinearAlgebra

# Model specification: y|x ~ 𝒩(f(x), v(x))
f(x) = 5*x .- 2
v(x) = 10*exp.(2*x.^2) .- 9.5 # input dependent noise variance
x_test = [0.0, 1.0]
plot(x_test, f(x_test), "k--") # plot f(x)

# Generate N samples (x,y), where x ~ Unif[0,1]
N = 50
x = rand(N)
y = f(x) + sqrt.(v(x)) .* randn(N)
plot(x, y, "kx"); xlabel("x"); ylabel("y") # Plot samples

# Add constant to input so we can estimate both the offset and the slope
_x = [x ones(N)]
_x_test = hcat(x_test, ones(2))

# LS regression
w_ls = pinv(_x) * y
plot(x_test, _x_test*w_ls, "b-") # plot LS solution

# Weighted LS regression
W = Diagonal(1 ./ v(x)) # weight matrix
w_wls = inv(_x'*W*_x) * _x' * W * y
plot(x_test, _x_test*w_wls, "r-") # plot WLS solution
ylim([-5,8]); legend(["f(x)", "D", "LS linear regr.", "WLS linear regr."],loc=2);
```


f(x)
D
LS linear regr.
WLS linear regr.
8
6
4
2
0
−2
−4
y
0.0
0.2
0.4
0.6
0.8
1.0
x

Generative Classification

Preliminaries

- Goal
  - Introduction to linear generative classification with a multinomial-Gaussian generative model
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 196-202 (section 4.2 focusses on binary classification, whereas in these lecture notes we describe generative classification for multiple classes).

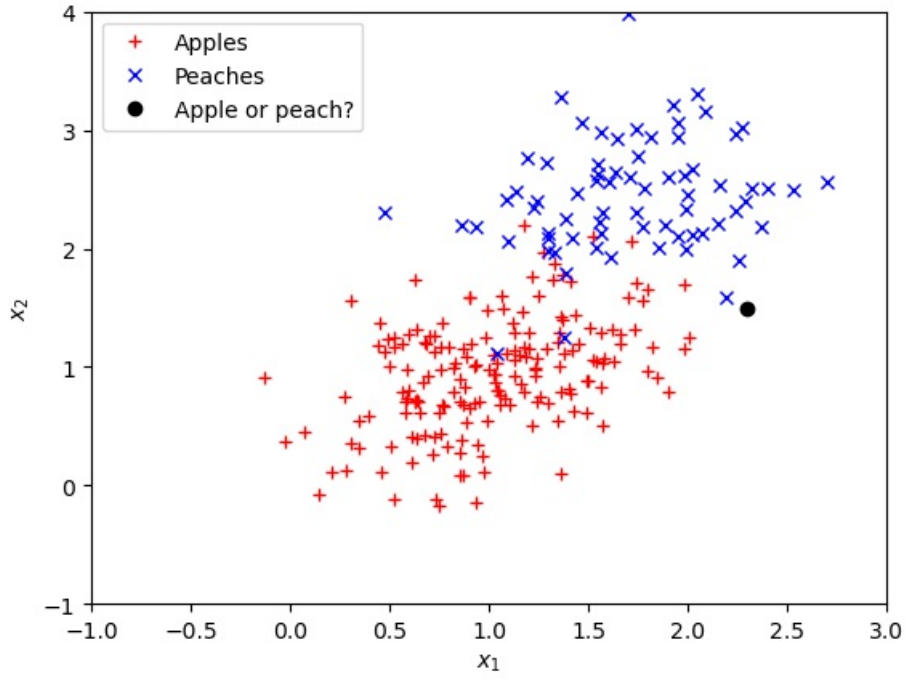
Challenge: an apple or a peach?

- Problem: You're given numerical values for the skin features roughness and color for 200 pieces of fruit, where for each piece of fruit you also know if it is an apple or a peach. Now you receive the roughness and color values for a new piece of fruit but you don't get its class label (apple or peach). What is the probability that the new piece is an apple?

- Solution: To be solved later in this lesson.
- Let's first generate a data set (see next slide).

```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate();
IJulia.clear_output();
```

```
using Distributions, PyPlot
N = 250; p_apple = 0.7; Σ = [0.2 0.1; 0.1 0.3]
p_given_apple = MvNormal([1.0, 1.0], Σ) # p(X|y=apple)
p_given_peach = MvNormal([1.7, 2.5], Σ) # p(X|y=peach)
X = Matrix{Float64}(undef,2,N); y = Vector{Bool}(undef,N) # true corresponds to apple
for n=1:N
    y[n] = (rand() < p_apple) # Apple or peach?
    X[:,n] = y[n] ? rand(p_given_apple) : rand(p_given_peach) # Sample features
end
X_apples = X[:,findall(y)]'; X_peaches = X[:,findall(.!y)]' # Sort features on class
x_test = [2.3; 1.5] # Features of 'new' data point

function plot_fruit_dataset()
    # Plot the data set and x_test
    plot(X_apples[:,1], X_apples[:,2], "r+")   # apples
    plot(X_peaches[:,1], X_peaches[:,2], "bx") # peaches
    plot(x_test[1], x_test[2], "ko")           # 'new' unlabelled data point
    legend(["Apples"; "Peaches"; "Apple or peach?"], loc=2)
    xlabel(L"x_1"); ylabel(L"x_2"); xlim([-1,3]); ylim([-1,4])
end
plot_fruit_dataset();
```



Generative Classification Problem Statement

- Given is a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - inputs $x_n \in \mathrm{R}^M$ are called features.
  - outputs $y_n \in$ $_k$, with $k = 1, \ldots, K$; The discrete targets $_k$ are called classes.

- We will again use the 1-of-K notation for the discrete classes. Define the binary class selection variable

$$y_{nk} = \begin{cases} 1 & \text{if } y_n \in \ _k \\ 0 & \text{otherwise} \end{cases}$$

  - (Hence, the notations $y_{nk} = 1$ and $y_n \in \ _k$ mean the same thing.)

- The plan for generative classification: build a model for the joint pdf $p(x,y) = p(x|y)p(y)$ and use Bayes to infer the posterior class probabilities

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_{y'} p(x|y')p(y')} \propto p(x|y)p(y)$$

1 – Model specification

Likelihood

- Assume Gaussian class-conditional distributions with equal covariance matrix across the classes,

$$p(x_n| \ _k) = \ (x_n|\mu_k, \Sigma)$$

  with notational shorthand: $_k \triangleq (y_n \in \ _k)$.

Prior

- We use a categorical distribution for the class labels y_{nk}:

$$p(\ _k) = \pi_k$$

- Hence, using the one-hot coding formulation for y_{nk}, the generative model $p(x_n, y_n)$ can be written as

$$\begin{aligned} p(x_n, y_n) &= \prod_{k=1}^{K} p(x_n, y_{nk} = 1)^{y_{nk}} \\ &= \prod_{k=1}^{K} \left(\pi_k \cdot \ (x_n|\mu_k, \Sigma)\right)^{y_{nk}} \end{aligned}$$

- In the literature, this model (with possibly unequal Σ_k across classes) is often called the Gaussian Discriminant Analysis (GDA) model.
  - The special case with equal covariance matrices $\Sigma_k = \Sigma$ is also called Linear Discriminant Analysis.

Computing the log-likelihood

- The log-likelihood given the full data set is then

$$\begin{aligned}\log p(D \mid \theta) &\overset{\text{IID}}{=} \sum_n \log \prod_k p(x_n, y_{nk} = 1 \mid \theta)^{y_{nk}} \\ &= \sum_{n,k} y_{nk} \log p(x_n, y_{nk} = 1 \mid \theta) \\ &= \sum_{n,k} y_{nk} \log p(x_n \mid y_{nk} = 1) + \sum_{n,k} y_{nk} \log p(y_{nk} = 1) \\ &= \sum_{n,k} y_{nk} \log \quad (x_n \mid \mu_k, \Sigma) + \sum_{n,k} y_{nk} \log \pi_k \\ &= \sum_{n,k} y_{nk} \log \quad (x_n \mid \mu_k, \Sigma) + \sum_k m_k \log \pi_k\end{aligned}$$

(see Gaussian est.; see multinomial est.)

  where we used $m_k \triangleq \sum_n y_{nk}$.

- As usual, the rest (inference for parameters and model prediction) through straight probability theory.

2 - Parameter Inference for Classification

- We'll do Maximum Likelihood estimation for $\theta = \{\pi_k, \mu_k, \Sigma\}$ from data D

- Recall (from the previous slide) the log-likelihood (LLH)

$$\log p(D \mid \theta) = \sum_{n,k} y_{nk} \log \quad (x_n \mid \mu_k, \Sigma) + \sum_k m_k \log \pi_k$$

(Gaussian; multinomial)

- Maximization of the LLH for the GDA model breaks down into
  - Gaussian density estimation for parameters μ_k, Σ, since the first term contains exactly the log-likelihood for MVG density estimation. We've already done this, see the Gaussian distribution lesson.
  - Multinomial density estimation for class priors π_k, since the second term holds exactly the log-likelihood for multinomial density estimation, see the Multinomial distribution lesson.

- The ML for multinomial class prior (we've done this before!)

$$\hat{\pi}_k = \frac{m_k}{N}$$

- Now group the data into separate classes and do MVG ML estimation for class-conditional parameters (we've done this before as well):

$$\hat{\mu}_k = \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}} = \frac{1}{m_k} \sum_n y_{nk} x_n$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{n,k} y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T$$

$$= \sum_k \hat{\pi}_k \cdot \left(\frac{1}{m_k} \sum_n y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \right)$$

$$\text{class-cond. variance}$$

$$= \sum_k \hat{\pi}_k \cdot \hat{\Sigma}_k$$

where $\hat{\pi}_k$, $\hat{\mu}_k$ and $\hat{\Sigma}_k$ are the sample proportion, sample mean and sample variance for the kth class, respectively.

- Note that the binary class selection variable y_{nk} groups data from the same class.

3 - Application: Class prediction for new Data

- Let's apply the trained model to predict the class for given a 'new' input $x_.$:

$$p(\ _k | x_., D) = \int p(\ _k | x_., \theta) p(\theta | D) \quad \mathrm{d}\theta$$

$$\text{ML: } \delta(\theta - \hat{\theta})$$

$$= p(\ _k | x_., \hat{\theta})$$

$$\propto p(\ _k) p(x_. |\ _k)$$

$$= \hat{\pi}_k \cdot \quad (x_. | \hat{\mu}_k, \hat{\Sigma})$$

$$\propto \hat{\pi}_k \exp\left\{ -\frac{1}{2} (x_. - \hat{\mu}_k)^T \hat{\Sigma}^{-1} (x_. - \hat{\mu}_k) \right\}$$

$$\propto \exp\left\{ \hat{\mu}_k^T \hat{\Sigma}^{-1} x_. - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k \right\}$$

$$\propto \frac{1}{Z} \exp\{\beta_k^T x_. + \gamma_k\}$$

where

$$\beta_k = \hat{\Sigma}^{-1} \hat{\mu}_k$$

$$\gamma_k = -\frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k$$

$$Z = \sum_{k'} \exp\{\beta_{k'}^T x + \gamma_{k'}\}. \quad \text{(normalization)}$$

- The class posterior function

$$\phi(a_k) \triangleq \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$

is called a softmax function. Note that the softmax function is per definition properly normalized in the sense that $\sum_k \phi(a_k) = 1$.

- The softmax function is a smooth approximation to the max-function. Note that we did not a priori specify a softmax posterior, but rather it followed from applying Bayes rule to the prior and likelihood assumptions.

Discrimination Boundaries

- The class log-posterior $\log p(\ _k | x) \propto \beta_k^T x + \gamma_k$ is a linear function of the input features.

- Thus, the contours of equal probability (discriminant functions) are lines (hyperplanes) in the feature space

$$\log \frac{p(\ _k | x, \theta)}{p(\ _j | x, \theta)} = \beta_{kj}^T x + \gamma_{kj} = 0$$

where we defined $\beta_{kj} \triangleq \beta_k - \beta_j$ and similarly for γ_{kj}.

- How to classify a new input $x_.$? The Bayesian answer is a posterior distribution $p(\ _k | x_.)$. If you must choose, then the class with maximum posterior class probability

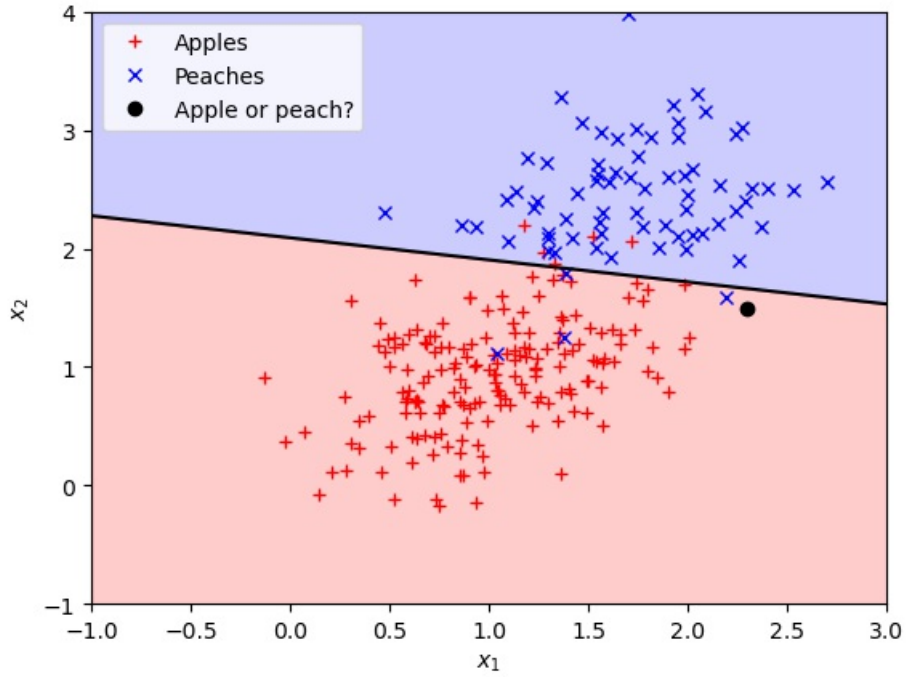
$$\begin{aligned} k^* &= \arg\max_k p(\ _k | x_.) \\ &= \arg\max_k \left(\beta_k^T x_. + \gamma_k \right) \end{aligned}$$

is an appealing decision.

CODE EXAMPLE

We'll apply the above results to solve the "apple or peach" example problem.

```
# Make sure you run the data-generating code cell first

# Multinomial (in this case binomial) density estimation
p_apple_est = sum(y.==true) / length(y)
π_hat = [p_apple_est; 1-p_apple_est]

# Estimate class-conditional multivariate Gaussian densities
d1 = fit_mle(FullNormal, X_apples')  # MLE density estimation d1 = N(μ₁, Σ₁)
d2 = fit_mle(FullNormal, X_peaches') # MLE density estimation d2 = N(μ₂, Σ₂)
Σ = π_hat[1]*cov(d1) + π_hat[2]*cov(d2) # Combine Σ₁ and Σ₂ into Σ
conditionals = [MvNormal(mean(d1), Σ); MvNormal(mean(d2), Σ)] # p(x|C)

# Calculate posterior class probability of x· (prediction)
function predict_class(k, X) # calculate p(Ck|X)
    norm = π_hat[1]*pdf(conditionals[1],X) + π_hat[2]*pdf(conditionals[2],X)
    return π_hat[k]*pdf(conditionals[k], X) ./ norm
end
println("p(apple|x=x·) = $(predict_class(1,x_test))")

# Discrimination boundary of the posterior (p(apple|x;D) = p(peach|x;D) = 0.5)
β(k) = inv(Σ)*mean(conditionals[k])
γ(k) = -0.5 * mean(conditionals[k])' * inv(Σ) * mean(conditionals[k]) + log(π_hat[k])
function discriminant_x2(x1)
    # Solve discriminant equation for x2
    β12 = β(1) .- β(2)
    γ12 = (γ(1) .- γ(2))[1,1]
    return -1*(β12[1]*x1 .+ γ12) ./ β12[2]
end

plot_fruit_dataset() # Plot dataset
x1 = range(-1,length=10,stop=3)
plot(x1, discriminant_x2(x1), "k-") # Plot discrimination boundary
fill_between(x1, -1, discriminant_x2(x1), color="r", alpha=0.2)
fill_between(x1, discriminant_x2(x1), 4, color="b", alpha=0.2);
```




```
p(apple|x=x·) = 0.7403119807623664
```

Recap Generative Classification

- Gaussian Discriminant Analysis model specification:

$$p(x, \quad _k \mid \theta) = \pi_k \cdot \quad (x \mid \mu_k, \Sigma)$$

- If the class-conditional distributions are Gaussian with equal covariance matrices across classes ($\Sigma_k = \Sigma$), then the discriminant functions are hyperplanes in feature space.

- ML estimation for $\{\pi_k, \mu_k, \Sigma\}$ in the GDA model breaks down to simple density estimation for Gaussian and multinomial.

Discriminative Classification

Preliminaries

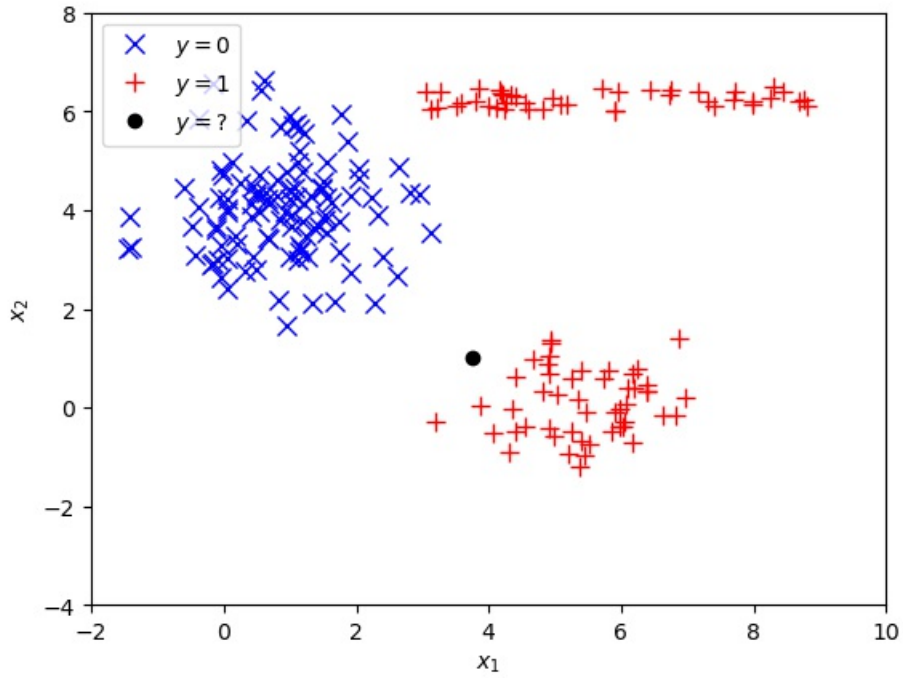
- Goal
  - Introduction to discriminative classification models
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 213 – 217 (Laplace approximation)
    - Bishop pp. 217 – 220 (Bayesian logistic regression)
    - T. Minka (2005), Discriminative models, not discriminative training

Challenge: difficult class-conditional data distributions

Our task will be the same as in the preceding class on (generative) classification. But this time, the class-conditional data distributions look very non-Gaussian, yet the linear discriminative boundary looks easy enough:

```
using Pkg;Pkg.activate("probprog/workspace");Pkg.instantiate()
using Random; Random.seed!(1234);
IJulia.clear_output();
```

```
# Generate dataset {(x1,y1),...,(xN,yN)}
# x is a 2-d feature vector [x_1;x_2]
# y ∈ {false,true} is a binary class label
# p(x|y) is multi-modal (mixture of uniform and Gaussian distributions)
using PyPlot
include("./scripts/lesson8_helpers.jl")
N = 200
X, y = genDataset(N) # Generate data set, collect in matrix X and vector y
X_c1 = X[:,findall(.!y)]'; X_c2 = X[:,findall(y)]' # Split X based on class label
X_test = [3.75; 1.0] # Features of 'new' data point
function plotDataSet()
    plot(X_c1[:,1], X_c1[:,2], "bx", markersize=8)
    plot(X_c2[:,1], X_c2[:,2], "r+", markersize=8, fillstyle="none")
    plot(X_test[1], X_test[2], "ko")
    xlabel(L"x_1"); ylabel(L"x_2");
    legend([L"y=0", L"y=1",L"y=?"], loc=2)
    xlim([-2;10]); ylim([-4, 8])
end
plotDataSet();
```



Main Idea of Discriminative Classification

- Again, a data set is given by $D = \{(x_1,y_1),\ldots,(x_N,y_N)\}$ with $x_n \in \mathrm{R}^M$ and $y_n \in \quad_k$, with $k = 1,\ldots,K$.

- Sometimes, the precise assumptions of the (multinomial-Gaussian) generative model

$$p(x_n,y_n \in \quad_k|\theta) = \pi_k \cdot \quad (x_n|\mu_k,\Sigma)$$

  clearly do not match the data distribution.

- Here's an IDEA! Let's model the posterior

$$p(y_n \in \quad_k|x_n)$$

  directly, without any assumptions on the class densities.

- Of course, this implies also that we build direct models for the discrimination boundaries, which are given by

$$\frac{p(y_n \in \quad_k | x_n)}{p(y_n \in \quad_j | x_n)} \stackrel{!}{=} 1$$

Bayesian Logistic Regression

- We will work this idea out for a 2-class problem. Assume a data set is given by $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathrm{R}^M$ and $y_n \in \{0, 1\}$.

Model Specification

- What model should we use for the posterior distribution $p(y_n \in \quad_k | x_n)$?

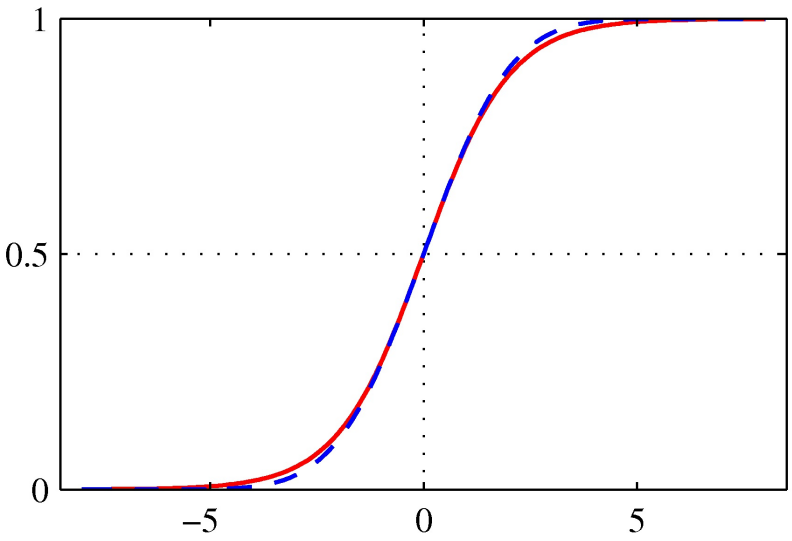
- In Logistic Regression, we take inspiration from the generative approach, where the softmax function "emerged" as the posterior. Here, we choose the 2-class softmax function (which is called the logistic function) with linear discrimination bounderies for the posterior class probability:

$$p(y_n = 1 \mid x_n, w) = \sigma(w^T x_n).$$

  where

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

  is the logistic function.



- (Bishop fig.4.9). The logistic function $\sigma(a) = 1/(1 + e^{-a})$ (red), together with the scaled probit function $\Phi(\lambda a)$, for $\lambda^2 = \pi/8$ (in blue). We will use this approximation later in the Laplace approximation.

- Indeed, for this choice of posterior class probabilities, the discrimination boundary is a straight line, see Exercises.

- Adding the other class ($y_n = 0$) leads to the following posterior class distribution:

$$\begin{aligned} p(y_n \mid x_n, w) &= \text{Bernoulli}\Big(y_n \mid \sigma(w^T x_n)\Big) \\ &= \sigma(w^T x_n)^{y_n}\Big(1 - \sigma(w^T x_n)\Big)^{(1-y_n)} \\ &= \sigma\Big((2y_n - 1)w^T x_n\Big) \end{aligned}$$

  - Note that for the 3rd equality, we have made use of the fact that $\sigma(-a) = 1 - \sigma(a)$.
  - Each of these three models in B-4.89 are equivalent. We mention all three notational options since they all appear in the literature.

- This choice for the class posterior is called logistic regression, in analogy to linear regression (where $p(y_n \mid x_n, w) = \quad (y_n \mid w^T x_n, \sigma^2)$).

- Note that in this model specification, we do not impose a Gaussian structure on the class features. In the discriminative approach, the parameters w are not structured into $\{\mu, \Sigma, \pi\}$. This provides discriminative approach with more flexibility than the generative approach.

- In Bayesian logistic regression, we add a Gaussian prior on the weights:

$$p(w) = \quad (w \mid m_0, S_0)$$

Inference

- The posterior for the weights follows by Bayes rule

$$\begin{aligned} p(w \mid D) &\propto p(w)p(D \mid w) \\ \text{posterior} \quad &= \quad (w \mid m_0, S_0) \cdot \prod_{n=1}^{N} \sigma\Big((2y_n - 1)w^T x_n\Big) \\ &\quad \text{prior} \qquad\qquad\qquad \text{likelihood} \end{aligned}$$

- In principle, Bayesian inference is done now. Unfortunately, the posterior is not Gaussian and the evidence $p(D)$ is also not analytically computable. (We will deal with this later).

Predictive distribution

- For a new data point $x_\bullet$, the predictive distribution for $y_\bullet$ is given by

$$\begin{aligned} p(y_\bullet = 1 \mid x_\bullet, D) &= \int p(y_\bullet = 1 \mid x_\bullet, w)p(w \mid D)\,\mathrm{d}w \\ &= \int \sigma(w^T x_\bullet)p(w \mid D)\,\mathrm{d}w \end{aligned}$$

- After substitution of $p(w|D)$ from B-4.142, we have an integral that is not solvable in closed-form.

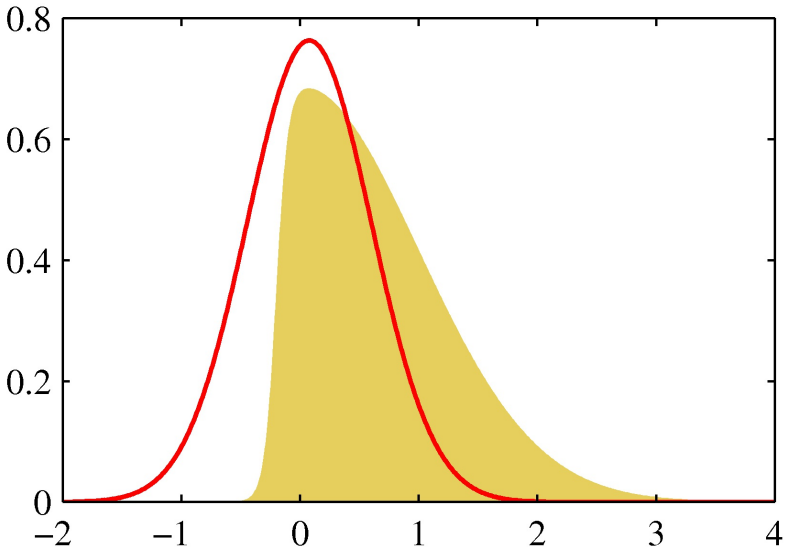
- Many methods have been developed to approximate the integrals for the predictive distribution and evidence. Here, we present the Laplace approximation, which is one of the simplest methods with broad applicability to Bayesian calculations.

The Laplace Approximation

- The central idea of the Laplace approximation is to approximate a (possibly unnormalized) distribution $f(z)$ by a Gaussian distribution $q(z)$.

- Note that $\log q(z)$ is a second-order polynomial in z, so we will find the Gaussian by fitting a parabola to $\log f(z)$.

Example



- (Bishop fig.4.14a). Laplace approximation (in red) to the distribution $p(z) \propto \exp(-z^2/2)\sigma(20z+4)$, where $\sigma(a) = 1/(1+e^{-a})$. The Laplace approximation is centered on the mode of $p(z)$.

Working out the Laplace Approximation

estimation of mean

- The mean (z_0) of $q(z)$ is placed on the mode of $\log f(z)$, i.e.,

$$z_0 = \arg\max_z \log f(z)$$

estimation of precision matrix

- Since the gradient $\nabla f(z)|_{z=z_0}$ vanishes at the mode, we can (Taylor) expand $\log f(z)$ around $z = z_0$ as

$$\log f(z) \approx \log f(z_0) + \underbrace{\left(\nabla \log f(z_0)\right)^T}_{=0 \text{ at } z=z_0}(z - z_0) + \ldots$$
$$+ \frac{1}{2}(z - z_0)^T\left(\nabla \nabla \log f(z_0)\right)(z - z_0)$$
$$= \log f(z_0) - \frac{1}{2}(z - z_0)^T A(z - z_0)$$

  where the Hessian matrix A is defined by

$$A = -\nabla \nabla \log f(z)|_{z=z_0}$$

Laplace approximation construction

- After taking exponentials in eq. B-4.131, we obtain

$$f(z) \approx f(z_0)\exp\left(-\frac{1}{2}(z - z_0)^T A(z - z_0)\right)$$

- We can now identify $q(z)$ as

$$q(z) = \left(z \mid z_0, A^{-1}\right)$$

  with z_0 and A defined by eqs. B-4.126 and B-4.132.

Bayesian Logistic Regression with the Laplace Approximation

- Let's get back to the challenge of computing the predictive class distribution (B-4.145) for Bayesian logistic regression. We first work out the Gaussian Laplace approximation $q(w)$ to the posterior weight distribution

$$\underbrace{p(w \mid D)}_{\text{posterior}} \propto \underbrace{(w \mid m_0, S_0)}_{\text{prior}} \cdot \underbrace{\prod_{n=1}^{N}\sigma\left((2y_n - 1)w^T x_n\right)}_{\text{likelihood}}$$

A Gausian Laplace approximation to the weights posterior

- Since we have a differentiable expression for $\log p(w \mid D)$, it is straightforward to compute the gradient and Hessian (for proof, see optional slide):

$$\nabla_w \log p(w \mid D) = S_0^{-1} \cdot \left(m_0 - w\right) + \sum_n (2y_n - 1)(1 - \sigma_n) x_n$$

$$\nabla \nabla_w \log p(w \mid D) = -S_0^{-1} - \sum_n \sigma_n (1 - \sigma_n) x_n x_n^T$$

  where we used shorthand σ_n for $\sigma\left((2y_n - 1) w^T x_n\right)$.

- We can now use the gradient $\nabla_w \log p(w \mid D)$ to find the mode w_N of $\log p(w \mid D)$ (eg by some gradient-based optimization procedure) and then use the Hessian $\nabla \nabla_w \log p(w \mid D)$ to get the variance of $q(w)$, leading to a **Gaussian approximate weights posterior**:

$$q(w) = \left(w \mid w_N, S_N\right)$$

  with

$$S_N^{-1} = S_0^{-1} + \sum_n \sigma_n (1 - \sigma_n) x_n x_n^T$$

Using the Laplace-approximated parameter posterior to evaluate the predictive distribution

- In the analytically unsolveable expressions for evidence and the predictive distribution (estimating the class of a new observation), we proceed with using the Laplace approximation to the weights posterior. For a new observation $x.$, the class probability is now

$$\begin{aligned} p(y. = 1 \mid x., D) &= \int p(y. = 1 \mid x., w) \cdot p(w \mid D) \, dw \\ &\approx \int p(y. = 1 \mid x., w) \cdot q(w) \quad dw \\ &\qquad\qquad\qquad\qquad \text{Gaussian} \\ &= \int \sigma(w^T x.) \cdot \quad \left(w \mid w_N, S_N\right) dw \end{aligned}$$

- This looks better but we need two more clever tricks to evaluate this expression.
    1. First, note that w appears in $\sigma(w^T x_\cdot)$ as an inner product, so through substitution of $a := w^T x_\cdot$, the expression simplifies to an integral over the scalar a (see Bishop for derivation):

$$p(y_\cdot = 1 \mid x_\cdot, D) \approx \int \sigma(a) \quad \left(a \mid \mu_a, \sigma_a^2\right) \mathrm{d}a$$
$$\mu_a = w_N^T x_\cdot$$
$$\sigma_a^2 = x_\cdot^T S_N x_\cdot$$

    2. Secondly, while the integral of the product of a logistic function with a Gaussian is not analytically solvable, the integral of the product of a Gaussian cumulative distribution function (CDF, also known as the probit function) with a Gaussian does have a closed-form solution. Fortunately,

$$\Phi(\lambda a) \approx \sigma(a)$$

    with the Gaussian CDF $\Phi(x) = \frac{1}{\sqrt{(2\pi)}} \int_{-\infty}^{x} e^{-t^2/2} \mathrm{d}t$, $\lambda^2 = \pi/8$ and $\sigma(a) = 1/(1+e^{-a})$. Thus, substituting $\Phi(\lambda a)$ with $\lambda^2 = \pi/8$ for $\sigma(a)$ leads to

$$p(y_\cdot = 1 \mid x_\cdot, D) = \int \sigma(w^T x_\cdot) \cdot p(w \mid D) \, \mathrm{d}w$$
$$\approx \int \underbrace{\Phi(\lambda a)}_{\text{probit function}} \cdot \underbrace{\left(a \mid \mu_a, \sigma_a^2\right) \mathrm{d}a}_{\text{Gaussian}}$$
$$= \Phi\left(\frac{\mu_a}{\sqrt{(\lambda^{-2} + \sigma_a^2)}}\right)$$

- We now have an approximate but closed-form expression for the predictive class distribution for a new observation with a Bayesian logistic regression model.

- Note that, by Eq.B-4.143, the variance S_N (and consequently σ_a^2) for the weight vector depends on the distribution of the training set. Large uncertainty about the weights (in areas with little training data and uninformative prior variance S_0) takes the posterior class probability eq. B-4.152 closer to 0.5. Does that make sense?

- Apparently, the Laplace approximation leads to a closed-form solutions for Bayesian logistic regression (although admittedly, the derivation is no walk in the park).

ML Estimation for Discriminative Classification

- Rather than the computationally involved Bayesian method, in practice, discriminative classification is often executed through maximum likelihood estimation.

- With the usual 1-of-K encoding scheme for classes ($y_{nk} = 1$ if $x_n \in$ $_k$, otherwise $y_{nk} = 0$), the log-likelihood for a K-dimensional discriminative classifier is

$$\begin{aligned} \mathrm{L}(\theta) &= \log \prod_n \prod_k p(\ _k \,|\, x_n, \theta)^{y_{nk}} \\ &= \log \prod_n \prod_k \left(\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}} \right)^{y_{nk}} \\ &\qquad \text{softmax function} \\ &= \sum_n \sum_k y_{kn} \log\left(\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}} \right) \end{aligned}$$

- Computing the gradient $\nabla_{\theta_k} \mathrm{L}(\theta)$ leads to (for proof, see optional slide below)

$$\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \underbrace{\left(y_{nk} - \frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}} \right)}_{\text{prediction error}} \cdot x_n$$

y_{nk}: target; $\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}$: prediction

- Compare this to the gradient for linear regression:

$$\nabla_\theta \mathrm{L}(\theta) = \sum_n \left(y_n - \theta^T x_n \right) x_n$$

- In both cases

$$\nabla_\theta \mathrm{L} = \sum_n \left(\text{target}_n - \text{prediction}_n \right) \cdot \text{input}_n$$

- The parameter vector θ for logistic regression can be estimated through iterative gradient-based adaptation. E.g. (with iteration index i),

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} + \eta \cdot \nabla_\theta \mathrm{L}(\theta) \Big|_{\theta = \hat{\theta}^{(i)}}$$

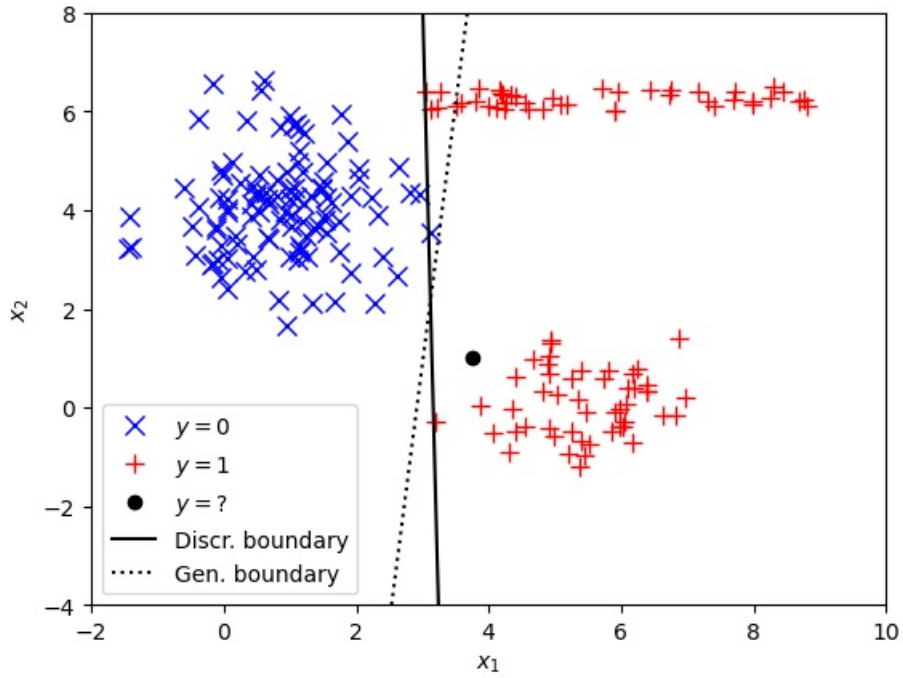
  - Note that, while in the Bayesian approach we get to update θ with precision-weighted prediction errors (which is optimal), in the maximum likelihood approach, we weigh the prediction errors with input values (which is less precise).

CODE EXAMPLE

Let us perform ML estimation of w on the data set from the introduction. To allow an offset in the discrimination boundary, we add a constant 1 to the feature vector x. We only have to specify the (negative) log-likelihood and the gradient w.r.t. w. Then, we use an off-the-shelf optimisation library to minimize the negative log-likelihood.

We plot the resulting maximum likelihood discrimination boundary. For comparison we also plot the ML discrimination boundary obtained from the code example in the generative Gaussian classifier lesson.

```
using Optim # Optimization library

y_1 = zeros(length(y))# class 1 indicator vector
y_1[findall(y)] .= 1
X_ext = vcat(X, ones(1, length(y))) # Extend X with a row of ones to allow an offset in the discri
mination boundary

# Implement negative log-likelihood function
function negative_log_likelihood(θ::Vector)
    # Return negative log-likelihood: -L(θ)
    p_1 = 1.0 ./ (1.0 .+ exp.(-X_ext' * θ))    # P(C1|X,θ)
    return -sum(log.( (y_1 .* p_1) + ((1 .- y_1).*(1 .- p_1))) ) # negative log-likelihood
end

# Use Optim.jl optimiser to minimize the negative log-likelihood function w.r.t. θ
results = optimize(negative_log_likelihood, zeros(3), LBFGS())
θ = results.minimizer

# Plot the data set and ML discrimination boundary
plotDataSet()
p_1(x) = 1.0 ./ (1.0 .+ exp(-([x;1.]' * θ)))
boundary(x1) = -1 ./ θ[2] * (θ[1]*x1 .+ θ[3])
plot([-2.;10.], boundary([-2.; 10.]), "k-");
# # Also fit the generative Gaussian model from lesson 7 and plot the resulting discrimination bou
ndary for comparison
generative_boundary = buildGenerativeDiscriminationBoundary(X, y)
plot([-2.;10.], generative_boundary([-2;10]), "k:");
legend([L"y=0";L"y=1";L"y=?";"Discr. boundary";"Gen. boundary"], loc=3);

# Given $\hat{\theta}$, we can classify a new input $x_\bullet = [3.75, 1.0]^T$:

x_test = [3.75;1.0]
println("P(C1|x•,θ) = $(p_1(x_test))")
```




```
P(C1|x•,θ) = 0.9999999132138306
```

- The generative model gives a bad result because the feature distribution of one class is clearly non-Gaussian: the model does not fit the data well.
- The discriminative approach does not suffer from this problem because it makes no assumptions about the feature distribition $p(x|y)$, it just estimates the conditional class distribution $p(y|x)$ directly.

Recap Classification

	Generative	Discriminative (ML)
	Like density estimation, model joint prob.	Like (linear) regression, model conditional
1	$p(\ _k)p(x\|\ _k) = \pi_k\ \ (\mu_k, \Sigma)$	$p(\ _k\|x, \theta)$
	Leads to softmax posterior class probability	Choose also softmax posterior class probability
2	$p(\ _k\|x, \theta) = e^{\theta_k^T x}/Z$	$p(\ _k\|x, \theta) = e^{\theta_k^T x}/Z$
	with structured θ	but now with 'free' θ
	For Gaussian $p(x\|\ _k)$ and multinomial priors,	
		Find $\hat{\theta}_k$ through gradient-based adaptation
3	$\hat{\theta}_k = \begin{bmatrix} -\frac{1}{2}\mu_k^T\sigma^{-1}\mu_k + \log\pi_k \\ \sigma^{-1}\mu_k \end{bmatrix}$	$\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \left(y_{nk} - \frac{e^{\theta_k^T x_n}}{\sum_{k'} e^{\theta_{k'}^T x_n}}\right) x_n$
	in one shot.	

OPTIONAL SLIDES

Proof of Derivative of Log-likelihood for Logistic Regression

- The Log-likelihood is $\mathrm{L}(\theta) = \log \prod_n \prod_k p(\ _k|x_n, \theta)^{y_{nk}} = \sum_{n,k} y_{nk} \log p_{nk}$

p_{nk}

- Use the fact that the softmax $\phi_k \equiv e^{a_k}/\sum_j e^{a_j}$ has analytical derivative:

$$\frac{\partial \phi_k}{\partial a_j} = \frac{(\sum_j e^{a_j}) e^{a_k}\delta_{kj} - e^{a_j}e^{a_k}}{(\sum_j e^{a_j})^2} = \frac{e^{a_k}}{\sum_j e^{a_j}}\delta_{kj} - \frac{e^{a_j}}{\sum_j e^{a_j}}\frac{e^{a_k}}{\sum_j e^{a_j}}$$
$$= \phi_k \cdot (\delta_{kj} - \phi_j)$$

- Take the derivative of $\mathrm{L}(\theta)$ (or: how to spend a hour ...)

$$\nabla_{\theta_j} \mathrm{L}(\theta) = \sum_{n,k} \frac{\partial \mathrm{L}_{nk}}{\partial p_{nk}} \cdot \frac{\partial p_{nk}}{\partial a_{nj}} \cdot \frac{\partial a_{nj}}{\partial \theta_j}$$
$$= \sum_{n,k} \frac{y_{nk}}{p_{nk}} \cdot p_{nk}(\delta_{kj} - p_{nj}) \cdot x_n$$
$$= \sum_n \left(y_{nj}(1 - p_{nj}) - \sum_{k \neq j} y_{nk} p_{nj}\right) \cdot x_n$$
$$= \sum_n \left(y_{nj} - p_{nj}\right) \cdot x_n$$
$$= \sum_n \left(y_{nj} - \frac{e^{\theta_j^T x_n}}{\sum_{j'} e^{\theta_{j'}^T x_n}}\right) \cdot x_n$$

target prediction

Proof of gradient and Hessian for Laplace Approximation of Posterior

- We will start with the posterior

$$\underset{\text{posterior}}{p(w \mid D)} \propto \underset{\text{prior}}{(w \mid m_0, S_0)} \cdot \underset{\text{likelihood}}{\prod_{n=1}^{N} \sigma\big((2y_n - 1) w^T x_n\big)}$$

  from which it follows that

$$\log p(w \mid D) \propto -\frac{1}{2}\log|S_0| - \frac{1}{2}(w - m_0)^T S_0^{-1}(w - m_0) + \sum_n \log\sigma\Big(a_n\Big)$$

  and the gradient

$$\nabla_w \log p(w \mid D) \propto \underset{\text{SRM-5b}}{S_0^{-1}(m_0 - w)} + \sum_n \underbrace{\frac{1}{\sigma(a_n)}}_{\frac{\partial \log\sigma(a_n)}{\partial\sigma(a_n)}} \cdot \underbrace{\sigma(a_n) \cdot (1 - \sigma(a_n))}_{\frac{\partial\sigma(a_n)}{\partial a_n}} \cdot \underbrace{(2y_n - 1)x_n}_{\frac{\partial a_n}{\partial w}\text{ (see SRM-5a)}}$$

$$= S_0^{-1}(m_0 - w) + \sum_n (2y_n - 1)(1 - \sigma(a_n))x_n \quad \text{(gradient)}$$

  where we used $\sigma'(a) = \sigma(a) \cdot (1 - \sigma(a))$.

- For the Hessian, we continue to differentiate the transpose of the gradient, leading to

$$\nabla\nabla_w \log p(w \mid D) = \nabla_w \Big(S_0^{-1}(m_0 - w)\Big)^T - \sum_n (2y_n - 1)x_n \nabla_w \sigma(a_n)^T$$

$$= -S_0^{-1} - \sum_n (2y_n - 1)x_n \cdot \underbrace{\sigma(a_n) \cdot (1 - \sigma(a_n))}_{\frac{\partial\sigma(a_n)^T}{\partial a_n^T}} \cdot \underbrace{(2y_n - 1)x_n^T}_{\frac{\partial a_n^T}{\partial w}}$$

$$= -S_0^{-1} - \sum_n \sigma(a_n) \cdot (1 - \sigma(a_n)) \cdot x_n x_n^T \quad \text{(Hessian)}$$

  since $(2y_n - 1)^2 = 1$ for $y_n \in \{0, 1\}$.

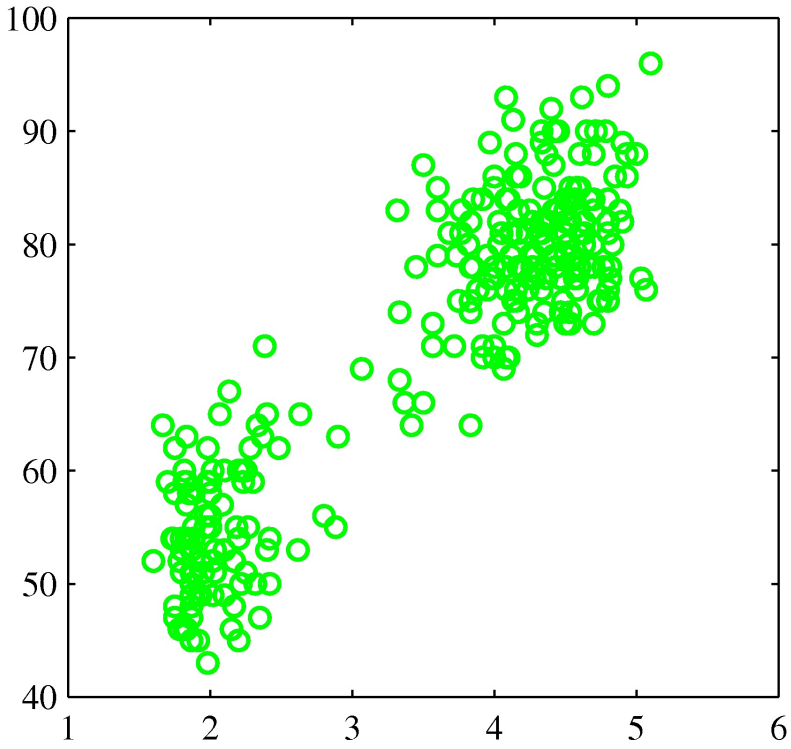
Latent Variable Models and Variational Bayes

Preliminaries

- Goal
  - Introduction to latent variable models and variational inference by Free energy minimization
- Materials
  - Mandatory
    - These lecture notes
    - Ariel Caticha (2010), Entropic Inference
      - tutorial on entropic inference, which is a generalization to Bayes rule and provides a foundation for variational inference.
  - Optional
    - Bishop (2016), pp. 461–486 (sections 10.1, 10.2 and 10.3)
  - references
    - Blei et al. (2017), Variational Inference: A Review for Statisticians
    - Lanczos (1961), The variational principles of mechanics
    - Zhang et al. (2017), Unifying Message Passing Algorithms Under the Framework of Constrained Bethe Free Energy Minimization
    - Dauwels (2007), On variational message passing on factor graphs
    - Caticha (2010), Entropic Inference
    - Shore and Johnson (1980), Axiomatic Derivation of the Principle of Maximum Entropy and the Principle of Minimum Cross-Entropy

Illustrative Example : Density Modeling for the Old Faithful Data Set

- You're now asked to build a density model for a data set (Old Faithful, Bishop pg. 681) that clearly is not distributed as a single Gaussian:



Unobserved Classes

Consider again a set of observed data $D = \{x_1, \ldots, x_N\}$

- This time we suspect that there are unobserved class labels that would help explain (or predict) the data, e.g.,
  - the observed data are the color of living things; the unobserved classes are animals and plants.
  - observed are wheel sizes; unobserved categories are trucks and personal cars.
  - observed is an audio signal; unobserved classes include speech, music, traffic noise, etc.

- Classification problems with unobserved classes are called Clustering problems. The learning algorithm needs to discover the classes from the observed data.

The Gaussian Mixture Model

- The spread of the data in the illustrative example looks like it could be modeled by two Gaussians. Let's develop a model for this data set.

- Let $D = \{x_n\}$ be a set of observations. We associate a one-hot coded hidden class label z_n with each observation:

$$z_{nk} = \begin{cases} 1 & \text{if } x_n \in \quad_k \text{ (the k-th class)} \\ 0 & \text{otherwise} \end{cases}$$

- We consider the same model as we did in the generative classification lesson:

$$p(x_n | z_{nk} = 1) = \quad \left(x_n | \mu_k, \Sigma_k\right)$$
$$p(z_{nk} = 1) = \pi_k$$

  which can be summarized with the selection variables z_{nk} as

$$p(x_n, z_n) = p(x_n | z_n) p(z_n) = \prod_{k=1}^{K} (\pi_k \cdot \quad \left(x_n | \mu_k, \Sigma_k\right))^{z_{nk}}$$
$$p(x_n, z_{nk} = 1)$$

- Again, this is the same model as we defined for the generative classification model: A Gaussian Discriminant Analysis model but now with unobserved classes).
- This model (with unobserved class labels) is known as a Gaussian Mixture Model (GMM).

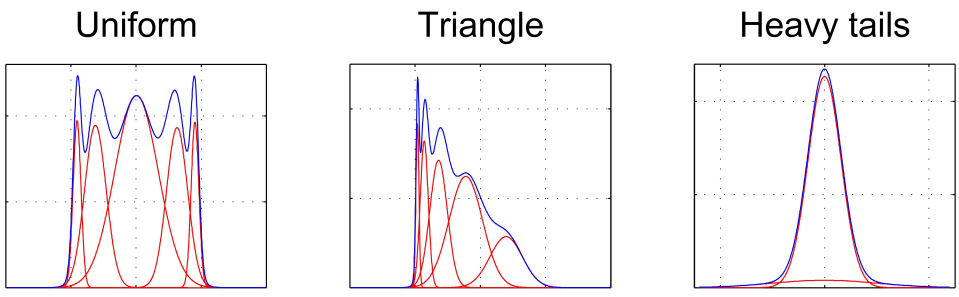
- The marginal distribution for an observed data point x_n is now given by

$$p(x_n) = \sum_{z_n} p(x_n, z_n)$$
$$= \sum_{k=1}^{K} \pi_k \cdot \quad \left(x_n | \mu_k, \Sigma_k\right)$$

- (Eq. B-9.12 reveals the link to the name Gaussian mixture model).

GMM is a very flexible model

- GMMs are very popular models. They have decent computational properties and are universal approximators of densities (as long as there are enough Gaussians of course)



- (In the above figure, the Gaussian components are shown in red and the pdf of the mixture models in blue).

Latent Variable Models

- The GMM contains both observed variables $\{x_n\}$, (unobserved) parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}$ and unobserved (synonym: latent, hidden) variables $\{z_{nk}\}$.

- From a Bayesian viewpoint, both latent variables $\{z_{nk}\}$ and parameters θ are just unobserved variables for which we can set a prior and compute a posterior by Bayes rule.

- Note that z_{nk} has a subscript n, hence its value depends not only on the cluster (k) but also on the n-th observation (in constrast to parameters). These observation-dependent variables are generally a useful tool to encode structure in the model. Here (in the GMM), the latent variables $\{z_{nk}\}$ encode (unobserved) class membership.

- By adding model structure through (equations among) latent variables, we can build very complex models. Unfortunately, inference in latent variable models can also be much more complex than for fully observed models.

Inference for GMM is Difficult

- We recall here the log-likelihood for the Gaussian Discriminative Analysis model (see generative classification lesson)

$$\log p(D \mid \theta) = \sum_{n,k} y_{nk} \log \underbrace{(x_n \mid \mu_k, \Sigma)}_{\text{Gaussian}} + \sum_{n,k} y_{nk} \log \underbrace{\pi_k}_{\text{multinomial}}.$$

- Since the class labels y_{nk} were given, this expression decomposed into a set of simple updates for the Gaussian and multinomial distributions. For both distributions, we have conjugate priors, so inference is easily solved.

- However, for the Gaussian mixture model (same log-likelihood function with z_{nk} replacing y_{nk}), the class labels $\{z_{nk}\}$ are unobserved and need to estimated alongside with the parameters.

- There is no conjugate prior for the GMM likelihood function $p(\underbrace{D}_{\{x_n\}} \mid \underbrace{\{z_{nk}\}}_{\text{latent variables}}, \{\mu_k, \Sigma_k, \pi_k\})$.

- In this lesson, we introduce an approximate Bayesian inference method known as Variational Bayes (VB) (also known as Variational Inference) that can be used for Bayesian inference in models with latent variables. Later in this lesson, we will use VB to do inference in the GMM.

- As a motivation for variational inference, we first discuss inference by the Method of Maximum Relative Entropy, (Caticha, 2010).

Inference When Information is in the Form of Constraints

- In the probability theory lesson, we recognized Bayes rule as the fundamental rule for learning from data.

- We will now generalize this notion and consider learning as a process that updates a prior into a posterior distribution whenever new information becomes available.

- In this context, new information is not necessarily a new observation, but could (for instance) also relate to constraints on the posterior distribution.

- For example, consider a model $p(x_n, \theta) = p(x_n|\theta)p(\theta)$. At this point, our beliefs about θ are represented by $p(\theta)$. We might be interested in obtaining rational posterior beliefs $q(\theta)$ in consideration of the following constraints:

  1. Data Constraints (observations): e.g., two new data points $x_1 = 5$ and $x_2 = 3$, which lead to constraints $q(x_1) = \delta(x_1 - 5)$ and $q(x_2) = \delta(x_2 - 3)$.
  2. Form Constraints: e.g., we only consider the Gamma distribution for $q(\theta) = \text{Gam}(\theta|\alpha, \beta)$.
  3. Moment Constraints: e.g., the first moment of the posterior is given by $\int \theta q(\theta) \text{d}\theta = 3$.
  4. other constraints

- Note that this is not "just" applying Bayes rule to compute a posterior, which can only deal with data constraints as specified by the likelihood function.

- Note also that observations can be rephrased as constraints on the posterior, e.g., observation $x_1 = 5$ can be phrased as a constraint $q(x_1) = \delta(x_1 - 5)$.
- $\Rightarrow$ Updating a prior to a posterior on the basis of constraints on the posterior is more general than updating based on observations alone.

- Caticha (2010) (based on earlier work by Shore and Johnson (1980)) developed the method of maximum (relative) entropy for rational updating of priors to posteriors when faced with new information in the form of constraints.

The Method of Maximum Relative Entropy

- Consider prior beliefs $p(z)$ about z. New information in the form of constraints is obtained and we are interested in the "best update" to a posterior $q(z)$.

- In order to define what "best update" means, we assume a ranking function $S[q,p]$ that generates a preference score for each candidate posterior q for a given p. The best update from p to q is identified as $q^* = \arg\max_q S[q,p]$.

- Similarly to Cox' method to deriving probability theory, Caticha introduced the following mild criteria based on a rational principle (the principle of minimal updating, see Caticha 2010) that the ranking function needs to adhere to:

  1. Locality: local information has local effects.
  2. Coordinate invariance: the system of coordinates carries no information.
  3. Independence: When systems are known to be independent, it should not matter whether they are treated separately or jointly.

- These three criteria uniquely identify the Relative Entropy (RE) as the proper ranking function:

$$S[q,p] = -\int q(z)\log\frac{q(z)}{p(z)}$$

- ⇒ When information is supplied in the form of constaints on the posterior, we should select the posterior that maximizes the Relative Entropy, subject to the constraints. This is the Method of Maximum (Relative) Entropy (MRE).

Relative Entropy, KL Divergence and Free Energy

- Note that the Relative Entropy is technically a functional, i.e., a function of function(s) (since it is a function of probability distributions). The calculus of functionals is also called variational calculus.
  - See Lanczos, The variational principles of mechanics (1961) for a great book on variational calculus. For a summary, see Bishop (2016), app.D.
  - It is customary to use square brackets (like $S[q,p]$) for functionals rather than round brackets, which we use for functions.

- Also note the complimentary relation between the Maximum Relative Entropy method and Probability Theory:
  - PT describes how to represent beliefs about events and how to calculate beliefs about joint and conditional events.
  - In contrast, the MRE method describes how to update beliefs when new information becomes available.

- PT and the MRE method are both needed to describe the theory of optimal information processing. (As we will see below, Bayes rule is a special case of the MRE method.)

- In principle, entropies can always be considered as a relative score against a reference distribution. For instance, the score $-\sum_{z_i} q(z_i) \log q(z_i)$ can be interpreted as a score against the uniform distribution, i.e., $-\sum_{z_i} q(z_i) \log q(z_i) \propto -\sum_{z_i} q(z_i) \log \frac{q(z_i)}{\text{Uniform}(z_i)}$. Therefore, the "method of maximum relative entropy" is often simply known as the "method of maximum entropy".

- The negative relative entropy is known as the Kullback-Leibler (KL) divergence:

$$\text{KL}[q,p] \triangleq \sum_z q(z) \log \frac{q(z)}{p(z)}$$

- The Gibbs inequality (proof), is a famous theorem in information theory that states that

$$\text{KL}[q,p] \geq 0$$

  with equality only iff $p = q$.

- The KL divergence can be interpreted as a "distance" between two probability distributions. Note however that the KL divergence is an asymmetric distance measure, i.e., in general $\text{KL}[q,p] \neq \text{KL}[p,q]$.

- We also introduce here the notion of a (variational) Free Energy (FE) functional, which is just a generalization of the KL-divergence that allows the prior to be unnormalized. Let $f(z) = Z \cdot p(z)$ be an unnormalized distribution, then the FE is defined as

$$\begin{aligned} F[q,p] &\triangleq \sum_z q(z) \log \frac{q(z)}{f(z)} \\ &= \sum_z q(z) \log \frac{q(z)}{Z \cdot p(z)} \\ &= \sum_z q(z) \log \frac{q(z)}{p(z)} - \log Z \end{aligned}$$

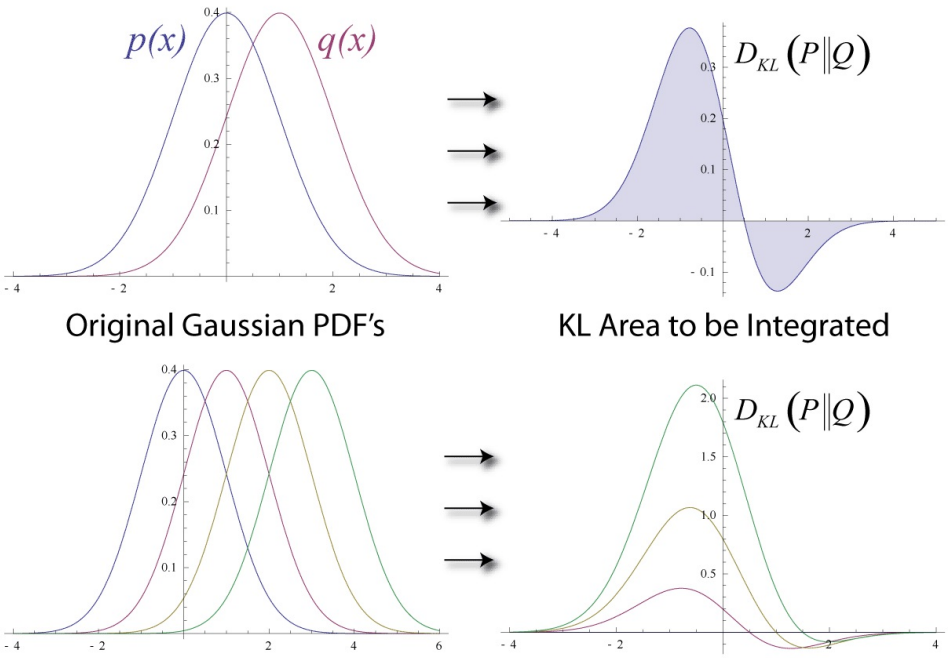
$$\text{KL divergence} \geq 0$$

- Since the second term ($\log Z$) is constant, FE minimization (subject to constraints) with respect to q leads to the same posteriors as KL minimization and RE maximization.

- If we are only interested in minimizing FE with respect to q, then we'll write $F[q]$ (rather than $F[q,p]$) fo brevity.

- In this class, we prefer to discuss inference in terms of minimizing Free Energy (subject to the constraints) rather than maximizing Relative Entropy, but note that these two concepts are equivalent.

Example KL divergence for Gaussians



source: By Mundhenk at English Wikipedia, CC BY-SA 3.0, Link

The Free Energy Functional and Variational Bayes

- Let's get back to the issue of doing inference for models with latent variables (such as the GMM).
- Consider a generative model specified by $p(x,z)$ where x and z represent the observed and unobserved variables, respectively.
- Assume further that x has been observed as $x = \hat{x}$ and we are interested in performing inference, i.e., we want to compute the posterior $p(z \mid x = \hat{x})$ for the latent variables and we want to compute the evidence $p(x = \hat{x})$.
- According to the Method of Maximum Relative Entropy, in order to find the correct posterior $q(x,z)$, we should minimize

$$\mathrm{F}[q] = \sum_{x,z} q(x,z) \log \frac{q(x,z)}{p(x,z)}, \quad \text{subject to data constraint } x = \hat{x}$$

- The data constraint $x = \hat{x}$ fixes posterior $q(x) = \delta(x - \hat{x})$, so the Free Energy evaluates to

$$\begin{aligned} F[q] &= \sum_z \sum_x q(z|x)q(x)\log\frac{q(z|x)q(x)}{p(z|x)p(x)} \\ &= \sum_z \sum_x q(z|x)\delta(x-\hat{x})\log\frac{q(z|x)\delta(x-\hat{x})}{p(z|x)p(x)} \\ &= \sum_z q(z|\hat{x})\log\frac{q(z|\hat{x})}{p(z|\hat{x})p(\hat{x})} \\ &= \underbrace{\sum_z q(z|\hat{x})\log\frac{q(z|\hat{x})}{p(z|\hat{x})}}_{\text{KL divergence} \geq 0} \underbrace{-\log p(\hat{x})}_{\text{log-evidence}} \end{aligned}$$

- The log-evidence term does not depend on q. Hence, the global minimum $q(z|\hat{x})^* \triangleq \arg\min_q F[q]$ is obtained for

$$q^*(z|\hat{x}) = p(z|\hat{x}),$$

  which is the correct Bayesian posterior.

- Furthermore, the minimal free energy

$$F^* \triangleq F[q^*] = -\log p(\hat{x})$$

  equals minus log-evidence. (Or, equivalently, the evidence is given by $p(\hat{x}) = \exp\left(-F[q^*]\right)$.)

⇒ Bayesian inference (computation of posterior and evidence in case there are only data constraints) can be executed by FE minimization.

- This is an amazing result! Note that FE minimization transforms an inference problem (that involves integration) to an optimization problem! Generally, optimization problems are easier to solve than integration problems.

- Executing inference by minimizing the variational FE functional is called Variational Bayes (VB) or variational inference.

- (As an aside), note that Bishop introduces in Eq. B-10.2 an Evidence Lower BOund (in modern machine learning literature abbreviated as ELBO) (z) that equals the negative FE. We prefer to discuss variational inference in terms of a free energy (but it is the same story as he discusses with the ELBO).

FE Minimization as Approximate Bayesian Inference

- In the rest of this lesson, we are concerned with how to execute FE minimization (FEM) w.r.t q for the functional

$$F[q] = \sum_z q(z)\log\frac{q(z)}{p(z\,|\,x)} - \log p(x)$$

  where x has been observed and z represent all latent variables.

  - To keep the notation uncluttered, in the following we write x rather than $\hat{x}$, and we write simply $q(z)$ (rather than $q(z|\hat{x})$) for the posterior.

- Due to restrictions in our optimization algorithm, we are often unable to fully minimize the FE to the global minimum $q^*(z)$, but rather get stuck in a local minimum $\hat{q}(z)$.

- Note that, since $\mathrm{KL}[q(z), p(z\,|\,x)] \geq 0$ for any $q(z)$, the FE is always an upperbound on (minus) log-evidence, i.e.,

$$F[q] \geq -\log p(x)\,.$$

- In practice, even if we cannot attain the global minimum, we can still use the local minimum $\hat{q}(z) \approx \arg\min_q F[q]$ to accomplish approximate Bayesian inference:

$$\text{posterior: } \hat{q}(z) \approx p(z\,|\,x)$$
$$\text{evidence: } p(x) \approx \exp\big(-F[\hat{q}]\big)$$

Constrained FE Minimization

- Generally speaking, it can be very helpful in the FE minimization task to add some additional constraints on $q(z)$ (i.e., in addition to the data constraints).

- Once we add constraints to q (in addition to data constraints), we are no longer guaranteed that the minimum of the (constrained) FE coincides with the solution by Bayes rule (which only takes data as constraints). So again, at best constrained FEM only an approximation to Bayes rule.

- There are three important cases of adding constraints to $q(z)$ that often alleviates the FE minimization task:

1. mean-field factorization

- We constrain the posterior to factorize into a set of independent factors, i.e.,

$$q(z) = \prod_{j=1}^{m} q_j(z_j),$$

2. fixed-form parameterization

- We constrain the posterior to be part of a parameterized probability distribution, e.g.,

$$q(z) = \quad (z \mid \mu, \Sigma).$$

  - In this case, the functional minimization problem for $F[q]$ reduces to the minimization of a function $F(\mu, \Sigma)$ w.r.t. its parameters.

3. the Expectation-Maximization (EM) algorithm

- We place some constraints both on the prior and posterior for z (to be discussed in the Optional Slides) that simplifies FE minimization to maximum-likelihood estimation.

FE Minimization with Mean-field Factorization Constraints: the CAVI Approach

- Let's work out FE minimization with additional mean-field constraints (=full factorization) constraints:

$$q(z) = \prod_{j=1}^{m} q_j(z_j).$$

  - In other words, the posteriors for z_j are all considered independent. This is a strong constraint but leads often to good solutions.

- Given the mean-field constraints, it is possible to derive the following expression for the optimal solutions $q_j^*(z_j)$, for $j = 1, \ldots, m$:

$$\log q_j^*(z_j) \propto \mathrm{E}_{q_{-j}^*}[\log p(x, z)]$$
$$= \sum_{z_{-j}} q_{-j}^*(z_{-j}) \log p(x, z)$$

"field"

"mean field"

where we defined $q_{-j}^*(z_{-j}) \triangleq q_1^*(z_1) q_2^*(z_2) \cdots q_{j-1}^*(z_{j-1}) q_{j+1}^*(z_{j+1}) \cdots q_m^*(z_m)$.

- Proof (from Blei, 2017): We first rewrite the energy-entropy decomposition of the FE as a function of $q_j(z_j)$ only:

$$F[q_j] = \mathrm{E}_{q_j}\Big[\mathrm{E}_{q_{-j}}\Big[\log p(x, z_j, z_{-j})\Big]\Big] - \mathrm{E}_{q_j}\Big[\log q_j(z_j)\Big] + \text{const.}\ ,$$

  where the constant holds all terms that do not depend on z_j. This expression can be written as

$$F[q_j] = \sum_{z_j} q_j(z_j) \log \frac{q_j(z_j)}{\exp\Big(\mathrm{E}_{q_{-j}}\Big[\log p(x, z_j, z_{-j})\Big]\Big)}$$

  which is a KL-divergence that is minimized by Eq. B-10.9.

- This is not yet a full solution to the FE minimization task since the solution $q_j^*(z_j)$ depends on expectations that involve other solutions $q_{i\neq j}^*(z_{i\neq j})$, and each of these other solutions $q_{i\neq j}^*(z_{i\neq j})$ depends on an expection that involves $q_j^*(z_j)$.

- In practice, we solve this chicken-and-egg problem by an iterative approach: we first initialize all $q_j(z_j)$ (for $j = 1, \ldots, m$) to an appropriate initial distribution and then cycle through the factors in turn by solving eq.B-10.9 and update $q_{-j}^*(z_{-j})$ with the latest estimates. (See Blei, 2017, Algorithm 1, p864).
- This algorithm for approximating Bayesian inference is known Coordinate Ascent Variational Inference (CAVI).

Example: FEM for the Gaussian Mixture Model (CAVI Approach)

- Let's get back to the illustrative example at the beginning of this lesson: we want to do density modeling for the Old Faithful data set.

`model specification`

- We consider a Gaussian Mixture Model, specified by

$$\begin{aligned} p(x, z \mid \theta) &= p(x \mid z, \mu, \Lambda) p(z \mid \pi) \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \cdot \Big(x_n \mid \mu_k, \Lambda_k^{-1}\Big)^{z_{nk}} \end{aligned}$$

  with tuning parameters $\theta = \{\pi_k, \mu_k, \Lambda_k\}$.

- Let us introduce some priors for the parameters. We factorize the prior and choose conjugate distributions by

$$p(\theta) = p(\pi, \mu, \Lambda) = p(\pi)p(\mu \mid \Lambda)p(\Lambda)$$

with

$$p(\pi) = \mathrm{Dir}(\pi \mid \alpha_0) = C(\alpha_0)\prod_k \pi_k^{\alpha_0 - 1}$$

$$p(\mu \mid \Lambda) = \prod_k \quad \left(\mu_k \mid m_0, \left(\beta_0 \Lambda_k\right)^{-1}\right)$$

$$p(\Lambda) = \prod_k \quad \left(\Lambda_k \mid W_0, \nu_0\right)$$

where $(\cdot)$ is a Wishart distribution (i.e., a multi-dimensional Gamma distribution).

- The full generative model is now specified by

$$p(x, z, \pi, \mu, \Lambda) = p(x \mid z, \mu, \Lambda)p(z \mid \pi)p(\pi)p(\mu \mid \Lambda)p(\Lambda)$$

with hyperparameters $\{\alpha_0, m_0, \beta_0, W_0, \nu_0\}$.

inference

- Assume that we have observed $D = \left\{x_1, x_2, \ldots, x_N\right\}$ and are interested to infer the posterior distribution for the tuning parameters:

$$p(\theta \mid D) = p(\pi, \mu, \Lambda \mid D)$$

- The (perfect) Bayesian solution is

$$p(\theta \mid D) = \frac{p(x = D, \theta)}{p(x = D)} = \frac{\sum_z p(x = D, z, \pi, \mu, \Lambda)}{\sum_z \sum_\pi \iint p(x = D, z, \pi, \mu, \Lambda)\, \mathrm{d}\mu \mathrm{d}\Lambda}$$

but this is intractable (See Blei (2017), p861, eqs. 8 and 9).

- Alternatively, we can use FE minimization with factorization constraint

$$q(\theta) = q(z) \cdot q(\pi, \mu, \Lambda)$$

on the posterior. For the specified model, this leads to FE minimization wrt the hyperparameters, i.e., we need to minimize the function $F(\alpha_0, m_0, \beta_0, W_0, \nu_0)$.

- Bishop shows that the equations for the optimal solutions (Eq. B-10.9) are analytically solvable for the GMM as specified above, leading to (for $k = 1, \ldots, K$):

$$\begin{aligned}
\alpha_k &= \alpha_0 + N_k \\
\beta_k &= \beta_0 + N_k \\
m_k &= \frac{1}{\beta_k}\left(\beta_0 m_0 + N_k \bar{x}_k\right) \\
W_k^{-1} &= W_0^{-1} + N_k S_k + \frac{\beta_0 N_k}{\beta_0 + N_k}\left(\bar{x}_k - m_0\right)\left(\bar{x}_k - m_0\right)^T \\
\nu_k &= \nu_0 + N_k
\end{aligned}$$

  where we used

$$\begin{aligned}
\log \rho_{nk} &= \mathrm{E}\left[\log \pi_k\right] + \frac{1}{2}\mathrm{E}\left[\log |\Lambda_k|\right] - \frac{D}{2}\log(2\pi) \\
&\quad - \frac{1}{2}\mathrm{E}\left[(x_k - \mu_k)^T \Lambda_k (x_k - \mu_k)\right] \\
r_{nk} &= \frac{\rho_{nk}}{\sum_{j=1}^{K} \rho_{nj}} \\
N_k &= \sum_{n=1}^{N} r_{nk} x_n \\
\bar{x}_k &= \frac{1}{N_k}\sum_{n=1}^{N} r_{nk} x_n \\
S_k &= \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}\left(x_n - \bar{x}_k\right)\left(x_n - \bar{x}_k\right)^T
\end{aligned}$$

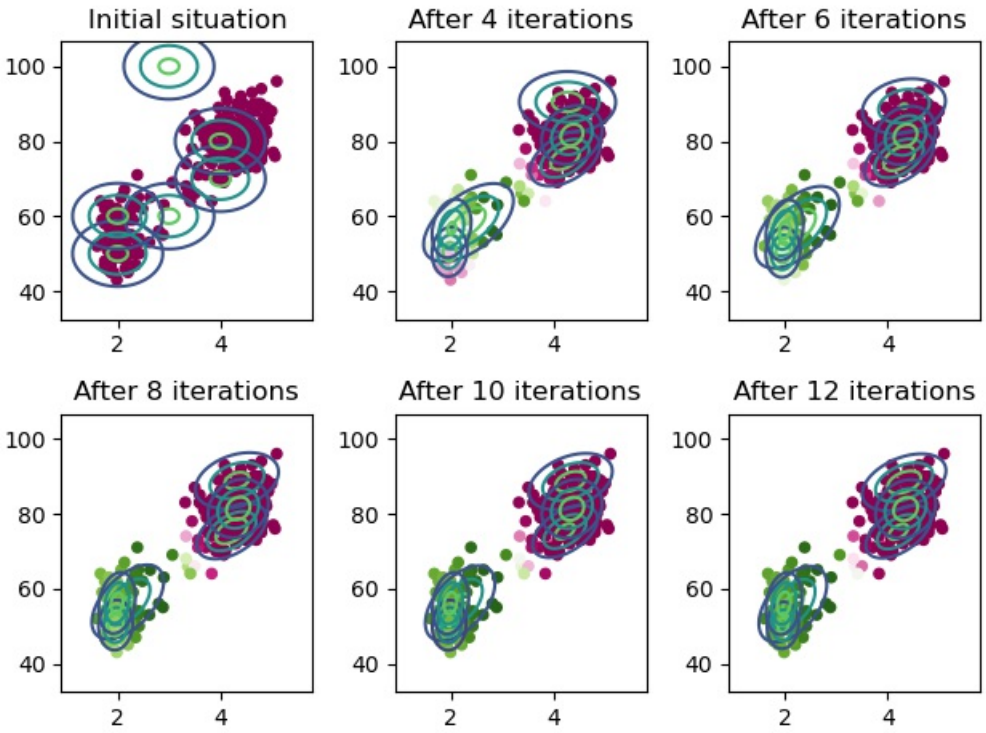
Code Example: FEM for GMM on Old Faithfull data set

- Below we exemplify training of a Gaussian Mixture Model on the Old Faithful data set by Free Energy Minimization, using the constraints as specified above, e.g., (B-10.42).

```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate()
IJulia.clear_output();
```

```
using DataFrames, CSV, LinearAlgebra, PDMats, SpecialFunctions
include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = CSV.read("datasets/old_faithful.csv");
X = convert(Matrix{Float64}, [old_faithful[!,1] old_faithful[!,2]]');#data matrix
N = size(X, 2) #number of observations
K = 6

function sufficientStatistics(X,r,k::Int) #function to compute sufficient statistics
    N_k = sum(r[k,:])
    hat_x_k = sum([r[k,n]*X[:,n] for n in 1:N]) ./ N_k
    S_k = sum([r[k,n]*(X[:,n]-hat_x_k)*(X[:,n]-hat_x_k)' for n in 1:N]) ./ N_k
    return N_k, hat_x_k, S_k
end

function updateMeanPrecisionPi(m_0,β_0,W_0,ν_0,α_0,r) #variational maximisation function
    m = Array{Float64}(undef,2,K) #mean of the clusters
    β = Array{Float64}(undef,K) #precision scaling for Gausian distribution
    W = Array{Float64}(undef,2,2,K) #precision prior for Wishart distributions
    ν = Array{Float64}(undef,K) #degrees of freedom parameter for Wishart distribution
    α = Array{Float64}(undef,K) #Dirichlet distribution parameter
```

```
    for k=1:K
        sst = sufficientStatistics(X,r,k)
        α[k] = α_0[k] + sst[1]; β[k] = β_0[k] + sst[1]; ν[k] = ν_0[k] .+ sst[1]
        m[:,k] = (1/β[k])*(β_0[k].*m_0[:,k] + sst[1].*sst[2])
        W[:,:,k] = inv(inv(W_0[:,:,k])+sst[3]*sst[1] + ((β_0[k]*sst[1])/(β_0[k]+sst[1])).*(sst[2]-
m_0[:,k])*(sst[2]-m_0[:,k])')
    end
    return m,β,W,ν,α
end

function updateR(Λ,m,α,ν,β) #variational expectation function
    r = Array{Float64}(undef,K,N) #responsibilities
    hat_π = Array{Float64}(undef,K)
    hat_Λ = Array{Float64}(undef,K)
    for k=1:K
        hat_Λ[k] = 1/2*(2*log(2)+logdet(Λ[:,:,k])+digamma(ν[k]/2)+digamma((ν[k]-1)/2))
        hat_π[k] = exp(digamma(α[k])-digamma(sum(α)))
        for n=1:N
           r[k,n] = hat_π[k]*exp(-hat_Λ[k]-1/β[k] - (ν[k]/2)*(X[:,n]-m[:,k])'*Λ[:,:,k]*(X[:,n]-m[:
,k]))
        end
    end
    for n=1:N
        r[:,n] = r[:,n]./ sum(r[:,n]) #normalize to ensure r represents probabilities
    end
    return r
end

max_iter = 15
#store the inference results in these vectors
ν = fill!(Array{Float64}(undef,K,max_iter),3)
β = fill!(Array{Float64}(undef,K,max_iter),1.0)
α = fill!(Array{Float64}(undef,K,max_iter),0.01)
R = Array{Float64}(undef,K,N,max_iter)
M = Array{Float64}(undef,2,K,max_iter)
Λ = Array{Float64}(undef,2,2,K,max_iter)
clusters_vb = Array{Distribution}(undef,K,max_iter) #clusters to be plotted
#initialize prior distribution parameters
M[:,:,1] = [[3.0; 60.0];[4.0; 70.0];[2.0; 50.0];[2.0; 60.0];[3.0; 100.0];[4.0; 80.0]]
for k=1:K
    Λ[:,:,k,1] = [1.0 0;0 0.01]
    R[k,:,1] = 1/(K)*ones(N)
    clusters_vb[k,1] = MvNormal(M[:,k,1],PDMats.PDMat(convert(Matrix,Hermitian(inv(ν[1,1].*Λ[:,:,k
,1])))))
end
#variational inference
for i=1:max_iter-1
    #variational expectation
    R[:,:,i+1] = updateR(Λ[:,:,:,i],M[:,:,i],α[:,i],ν[:,i],β[:,i])
    #variational minimisation
    M[:,:,i+1],β[:,i+1],Λ[:,:,:,i+1],ν[:,i+1],α[:,i+1] = updateMeanPrecisionPi(M[:,:,i],β[:,i],Λ[:
,:,:,i],ν[:,i],α[:,i],R[:,:,i+1])
    for k=1:K
        clusters_vb[k,i+1] = MvNormal(M[:,k,i+1],PDMats.PDMat(convert(Matrix,Hermitian(inv(ν[k,i+1
].*Λ[:,:,k,i+1])))))
    end
end


subplot(2,3,1); plotGMM(X, clusters_vb[:,1], R[:,:,1]); title("Initial situation")
for i=2:6
    subplot(2,3,i)
    plotGMM(X, clusters_vb[:,i*2], R[:,:,i*2]); title("After $(i*2) iterations")
end
PyPlot.tight_layout();
```



The generated figure looks much like Figure 10.6 in Bishop. The plots show results for Variational Bayesian mixture of K = 6 Gaussians applied to the Old Faithful data set, in which the ellipses denote the one standard-deviation density contours for each of the components, and the color coding of the data points reflects the "soft" class label assignments. Components whose expected mixing coefficient are numerically indistinguishable from zero are not plotted. Note that this procedure learns the number of classes (two), learns the class label for each observation, and learns the mean and variance for the Gaussian data distributions.

Free Energy Decompositions and Fixed-form Constraints

- The FE as specified above depends on the posterior $p(z|x)$ and evidence $p(x)$, both of which are not accessible since they are the objectives of Bayesian inference. Fortunately, we can rewrite the FE in computable terms.
- Making use of $p(x,z) = p(z|x)p(x) = p(x|z)p(z)$, the FE functional can be rewritten as

$$\begin{aligned} \mathrm{F}[q] &= \sum_z q(z)\log\frac{q(z)}{p(z|x)} - \log p(x) \\ &\qquad \text{KL divergence} \geq 0 \qquad\qquad \text{log-evidence} \\ &= -\sum_z q(z)\log p(x,z) - \sum_z q(z)\log\frac{1}{q(z)} \\ &\qquad \text{energy} \qquad\qquad \text{entropy} \\ &= \sum_z q(z)\log\frac{q(z)}{p(z)} - \sum_z q(z)\log p(x|z) \\ &\qquad \text{complexity} \qquad\qquad \text{accuracy} \end{aligned}$$

- These decompositions are very insightful (we will revisit them later) and we will label them respectively as divergence-evidence (DE), energy-entropy (EE), and complexity-accuracy (CA) decompositions.
- The CA decomposition makes use of the prior $p(z)$ and likelihood $p(x|z)$, both of which are selected by the engineer, so the FE can be evaluated with this decomposition!
- If we now assume some parametric form for $q(z)$, e.g. $q(z) = \quad (z \mid \mu, \Sigma)$, then the FE functional degenerates to a regular function $F(\mu, \Sigma)$. In principle, this function can be evaluated by the CA decomposition and minimized by standard (function) minimization methods.

Summary

- Latent variable models (LVM) contain a set of unobserved variables whose size grows with the number of observations.
- LVMs can model more complex phenomena than fully observed models, but inference in LVM models is usually not analytically solvable.
- The Free Energy (FE) functional transforms Bayesian inference computations (very large summations or integrals) to an optimization problem.
- Inference by minimizing FE, also known as variational inference, is fully consistent with the "Method of Maximum Relative Entropy", which is by design the rational way to update beliefs from priors to posteriors when new information becomes available. Thus, FE mimimization is a "correct" inference procedure that generalizes Bayes rule.

- In general, global FE minimization is an unsolved problem and finding good local minima is at the heart of current Bayesian technology research. Three simplifying constraints on the posterior $q(z)$ in the FE functional are currently popular in practical settings:
  - mean-field assumptions
  - assuming a parametric from for q
  - EM algorithm

- These constraints often makes FE minimization implementable at the price of obtaining approximately Bayesian inference results.

- The back-ends of Probabilistic Programming languages often contain lots of methods for constrained FE minimization.

OPTIONAL SLIDES

FE Minimization by the Expectation-Maximization (EM) Algorithm

- The EM algorithm is a special case of FE minimization that focusses on Maximum-Likelihood estimation for models with latent variables.
- Consider a model

$$p(x, z, \theta)$$

  with observations $x = \{x_n\}$, latent variables $z = \{z_n\}$ and parameters θ.
- We can write the following FE functional for this model:

$$F[q] = \sum_z \sum_\theta q(z)q(\theta)\log\frac{q(z)q(\theta)}{p(x, z, \theta)}$$

- The EM algorithm makes the following simplifying assumptions:
  1. The prior for the parameters is uninformative (uniform). This implies that

$$p(x, z, \theta) = p(x, z \mid \theta)p(\theta) \propto p(x, z \mid \theta)$$

  2. The posterior for the parameters is a delta function:

$$q(\theta) = \delta(\theta - \hat{\theta})$$

- Basically, these two assumptions turn FE minimization into maximum likelihood estimation for the parameters θ and the FE simplifies to

$$F[q, \theta] = \sum_z q(z)\log\frac{q(z)}{p(x, z \mid \theta)}$$

- The EM algorithm minimizes this FE by iterating (iteration counter: i) over

$$q^{(i)}(z)$$

$$\mathcal{L}^{(i)}(\theta) = \sum_z p(z \mid x, \theta^{(i-1)}) \log p(x, z \mid \theta) \qquad \text{the E-step}$$

$$\theta^{(i)} = \arg\max_\theta \mathcal{L}^{(i)}(\theta) \qquad \text{the M-step}$$

- These choices are optimal for the given FE functional. In order to see this, consider the two decompositions

$$F[q, \theta] = -\sum_z q(z) \log p(x, z \mid \theta) - \sum_z q(z) \log \frac{1}{q(z)}$$

$$\text{energy} \qquad \text{entropy}$$

$$= \sum_z q(z) \log \frac{q(z)}{p(z \mid x, \theta)} - \log p(x \mid \theta)$$

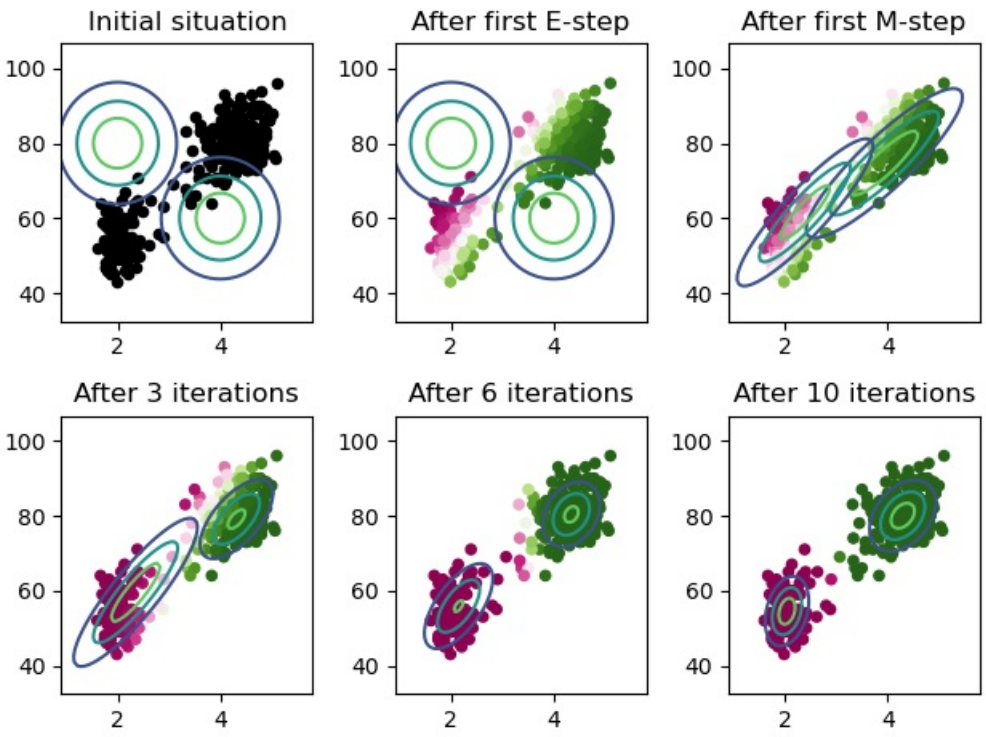
$$\text{divergence} \qquad \text{log-likelihood}$$

- The DE decomposition shows that the FE is minimized for the choice $q(z) := p(z \mid x, \theta)$. Also, for this choice, the FE equals the (negative) log-evidence (, which is this case simplifies to the log-likelihood).
- The EE decomposition shows that the FE is minimized wrt θ by minimizing the energy term. The energy term is computed in the E-step and optimized in the M-step.
  - Note that in the EM literature, the energy term is often called the expected complete-data log-likelihood.)
- In order to execute the EM algorithm, it is assumed that we can analytically execute the E- and M-steps. For a large set of models (including models whose distributions belong to the exponential family of distributions), this is indeed the case and hence the large popularity of the EM algorithm.
- The EM algorihm imposes rather severe assumptions on the FE (basically approximating Bayesian inference by maximum likelihood estimation). Over the past few years, the rise of Probabilistic Programming languages has dramatically increased the range of models for which the parameters can by estimated autmatically by (approximate) Bayesian inference, so the popularity of EM is slowly waning. (More on this in the Probabilistic Programming lessons).
- Bishop (2006) works out EM for the GMM in section 9.2.

CODE EXAMPLE: EM-algorithm for the GMM on the Old-Faithful data set

We'll perform clustering on the data set from the illustrative example by fitting a GMM consisting of two Gaussians using the EM algorithm.

```
using Pkg; Pkg.activate("probprog/workspace");Pkg.instantiate();
IJulia.clear_output();

using DataFrames, CSV, LinearAlgebra
include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = CSV.read("datasets/old_faithful.csv");

X =  Array(Matrix{Float64}(old_faithful)')
N = size(X, 2)

# Initialize the GMM. We assume 2 clusters.
clusters = [MvNormal([4.;60.], [.5 0;0 10^2]);
            MvNormal([2.;80.], [.5 0;0 10^2])];
π_hat = [0.5; 0.5]                    # Mixing weights
γ = fill!(Matrix{Float64}(undef,2,N), NaN)  # Responsibilities (row per cluster)

# Define functions for updating the parameters and responsibilities
function updateResponsibilities!(X, clusters, π_hat, γ)
    # Expectation step: update γ
    norm = [pdf(clusters[1], X) pdf(clusters[2], X)] * π_hat
    γ[1,:] = (π_hat[1] * pdf(clusters[1],X) ./ norm)'
    γ[2,:] = 1 .- γ[1,:]
end
function updateParameters!(X, clusters, π_hat, γ)
    # Maximization step: update π_hat and clusters using ML estimation
    m = sum(γ, dims=2)
    π_hat = m / N
    μ_hat = (X * γ') ./ m'
    for k=1:2
        Z = (X .- μ_hat[:,k])
        Σ_k = Symmetric(((Z .* (γ[k,:])') * Z') / m[k])
        clusters[k] = MvNormal(μ_hat[:,k], convert(Matrix, Σ_k))
    end
end

# Execute the algorithm: iteratively update parameters and responsibilities
subplot(2,3,1); plotGMM(X, clusters, γ); title("Initial situation")
updateResponsibilities!(X, clusters, π_hat, γ)
subplot(2,3,2); plotGMM(X, clusters, γ); title("After first E-step")
updateParameters!(X, clusters, π_hat, γ)
subplot(2,3,3); plotGMM(X, clusters, γ); title("After first M-step")
iter_counter = 1
for i=1:3
    for j=1:i+1
        updateResponsibilities!(X, clusters, π_hat, γ)
        updateParameters!(X, clusters, π_hat, γ)
        iter_counter += 1
    end
    subplot(2,3,3+i);
    plotGMM(X, clusters, γ);
    title("After $(iter_counter) iterations")
end
PyPlot.tight_layout()
```



Note that you can step through the interactive demo yourself by running this script in julia. You can run a script in julia by

```
julia> include("path/to/script-name.jl")
```

Message Passing for Free Energy Minimization

- The Sum-Product (SP) update rule implements perfect Bayesian inference.
- Sometimes, the SP update rule is not analytically solvable.
- Fortunately, for many well-known Bayesian approximation methods, a message passing update rule can be created, e.g. Variational Message Passing (VMP) for variational inference.
- In general, all of these message passing algorithms can be interpreted as minimization of a constrained free energy (e.g., see Zhang et al. (2017), and hence these message passing schemes comply with Caticha's Method of Maximum Relative Entropy, which, as discussed in the variational Bayes lesson is the proper way for updating beliefs.
- Different message passing updates rules can be combined to get a hybrid inference method in one model.

The Local Free Energy in a Factor Graph

- Consider an edge x_j in a Forney-style factor graph for a generative model $p(x) = p(x_1, x_2, \ldots, x_N)$.
- Assume that the graph structure (factorization) is specified by

$$p(x) = \prod_{a=1}^{M} p_a(x_a)$$

  where a is a set of indices.
- Also, we assume a mean-field approximation for the posterior:

$$q(x) = \prod_{i=1}^{N} q_i(x_i)$$

  and consequently a corresponding free energy functional

$$F[q] = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

$$= \sum_i \sum_{x_i} \left(\prod_{i=1}^{N} q_i(x_i) \right) \log \frac{\prod_{i=1}^{N} q_i(x_i)}{\prod_{a=1}^{M} p_a(x_a)}$$

- With these assumptions, it can be shown that the FE evaluates to (exercise)

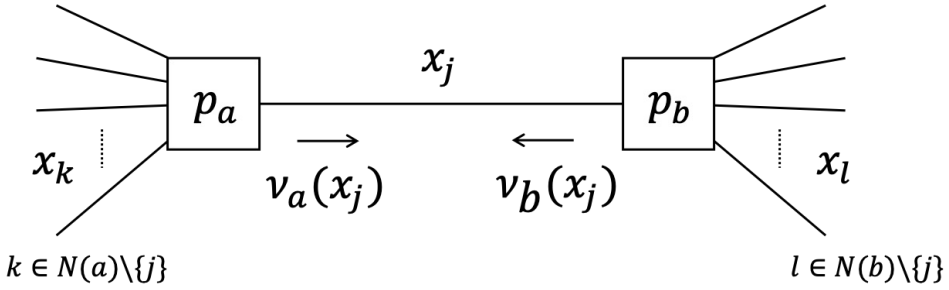
$$F[q] = \sum_{a=1}^{M} \sum_{x_a} \left(\prod_{j \in N(a)} q_j(x_j) \cdot \left(-\log p_a(x_a) \right) \right) - \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \log \frac{1}{q_i(x_i)}$$

  node energy $U[p_a]$ — edge entropy $H[q_i]$

- In words, the FE decomposes into a sum of (expected) energies for the nodes minus the entropies on the edges.

Variational Message Passing

- Let us now consider the local free energy that is associated with edge corresponding to x_j.



- Apparently (see previous slide), there are three contributions to the free energy for x_j:
  - one entropy term for the edge x_j
  - two energy terms: one for each node that attaches to x_j (in the figure: nodes p_a and p_b)
- The local free energy for x_j can be written as (exercise)

$$F[q_j] \propto \sum_{x_j} q(x_j) \log \frac{q_j(x_j)}{v_a(x_j) \cdot v_b(x_j)}$$

where

$$v_a(x_j) \propto \exp\left(\mathrm{E}_{q_k}\left[\log p_a(x_a)\right]\right)$$

$$v_b(x_j) \propto \exp\left(\mathrm{E}_{q_l}\left[\log p_b(x_b)\right]\right)$$

and $\mathrm{E}_{q_k}[\,\cdot\,]$ is an expectation w.r.t. all $q(x_k)$ with $k \in N(a) \setminus j$.

- $v_a(x_j)$ and $v_b(x_j)$ can be locally computed in nodes a and b respectively and can be interpreted as colliding messages over edge x_j.
- Local free energy minimization is achieved by setting

$$q_j(x_j) \propto v_a(x_j) \cdot v_b(x_j)$$

- Note that message $v_a(x_j)$ depends on posterior beliefs over incoming edges (k) for node a, and in turn, the message from node a towards edge x_k depends on the belief $q_j(x_j)$. I.o.w., direct mutual dependencies exist between posterior beliefs over edges that attach to the same node.
- These considerations lead to the Variational Message Passing procedure, which is an iterative free energy minimization procedure that can be executed completely through locally computable messages.
- Procedure VMP, see Dauwels (2007), section 3

1. Initialize all messages q and ν, e.g., $q(\,\cdot\,) \propto 1$ and $\nu(\,\cdot\,) \propto 1$.
2. Select an edge z_k in the factor graph of $f(z_1, \ldots, z_m)$.
3. Compute the two messages $\vec{\nu}(z_k)$ and $\overleftarrow{\nu}(z_k)$ by applying the following generic rule:

$$\vec{\nu}(y) \propto \exp\left(\mathrm{E}_q\left[\log g(x_1, \ldots, x_n, y)\right]\right)$$

4. Compute the marginal $q(z_k)$

$$q(z_k) \propto \vec{\nu}(z_k)\, \overleftarrow{\nu}(z_k)$$

   and send it to the two nodes connected to the edge x_k.
5. Iterate 2–4 until convergence.

The Bethe Free Energy and Belief Propagation

- We showed that, under mean field assumptions, the FE can be decomposed into a sum of local FE contributions for the nodes (a) and edges (i):

$$F[q] = \sum_{a=1}^{M} \sum_{x_a} \left(\prod_{j \in N(a)} q_j(x_j) \cdot \left(-\log p_a(x_a) \right) \right) - \sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \log \frac{1}{q_i(x_i)}$$

$$\text{node energy } U[p_a] \qquad\qquad \text{edge entropy } H[q_i]$$

- The mean field assumption is very strong and may lead to large inference costs ($\mathrm{KL}(q(x), p(x \mid \text{data}))$). A more relaxed assumption is to allow joint posterior beliefs over the variables that attach to a node. This idea is expressed by the Bethe Free Energy:

$$F_B[q] = \sum_{a=1}^{M} \left(\sum_{x_a} q_a(x_a) \log \frac{q_a(x_a)}{p_a(x_a)} \right) - \sum_{i=1}^{N} (d_i - 1) \sum_{x_i} q_i(x_i) \log q_i(x_i)$$

  where $q_a(x_a)$ is the posterior joint belief over the variables x_a (i.e., the set of variables that attach to node a), $q_i(x_i)$ is the posterior marginal belief over the variable x_i and d_i is the number of factor nodes that link to edge i. Moreover, $q_a(x_a)$ and $q_i(x_i)$ are constrained to obey the following equalities:

$$\sum_{x_a \setminus x_i} q_a(x_a) = q_i(x_i), \quad \forall i, \forall a \sum_{x_i} q_i(x_i) = 1, \quad \forall i$$

- Given these constraints, we form the Lagrangian by augmenting the Bethe Free Energy functional with the constraints

$$L[q] = F_B[q] + \sum_{a,i} \lambda_{ai}(x_i) \left(q_i(x_i) - \sum_{x_a \setminus x_i} q(x_a) \right) + \sum_i \lambda_i \left(1 - \sum_{x_i} q_i(x_i) \right)$$

- The stationary solutions for this Lagrangian are given by

$$q_a(x_a) \propto f_a(x_a) \exp\left(\sum_{i \in N(a)} \lambda_{ai}(x_i) \right) q_i(x_i) \propto \exp\left(\sum_{a \in N(i)} \lambda_{ai}(x_i) \right)^{\frac{1}{d_i - 1}}$$

  where $N(i)$ denotes the factor nodes that have x_i in their arguments and $N(a)$ denotes the set of variables in the argument of f_a.

- TO BE FINISHED ...

- For a more complete overview of message passing as Bethe Free Energy minimization, see Zhang (2017).

Dynamic Models

Preliminaries

- Goal
  - Introduction to dynamic (=temporal) Latent Variable Models, including the Hidden Markov Model and Kalman filter.
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp.605-615 on Hidden Markov Models
    - Bishop pp.635-641 on Kalman filters
    - Faragher (2012), Understanding the Basis of the Kalman Filter
    - Minka (1999), From Hidden Markov Models to Linear Dynamical Systems

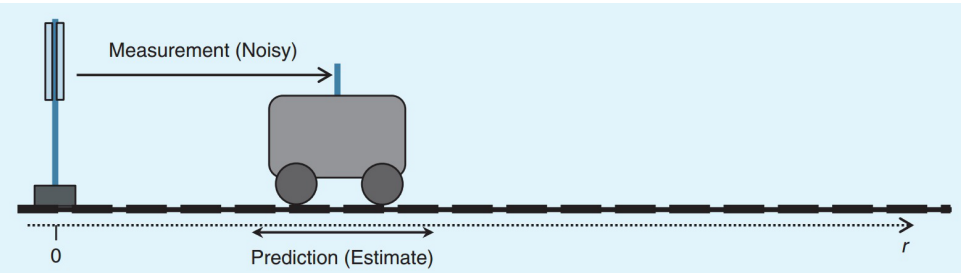
Example Problem

- We consider a one-dimensional cart position tracking problem, see Faragher 2012.
- The hidden states are the position z_t and velocity $\dot{z}_t$. We can apply an external acceleration/breaking force u_t. (Noisy) observations are represented by x_t.
- The equations of motions are given by

$$\begin{bmatrix} z_t \\ \cdot \\ z_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \quad (0, \Sigma_z)$$

$$x_t = \begin{bmatrix} z_t \\ \cdot \\ z_t \end{bmatrix} + \quad (0, \Sigma_x)$$

- Task: Infer the position z_t after 10 time steps. (Solution later in this lesson).



Dynamical Models

- In this lesson, we consider models where the sequence order of observations matters.
- Consider the ordered observation sequence $x^T \triangleq \left(x_1, x_2, \ldots, x_T\right)$.
  - (For brevity, in this lesson we use the notation x_t^T to denote $(x_t, x_{t+1}, \ldots, x_T)$ and drop the subscript if $t = 1$, so $x^T = x_1^T = \left(x_1, x_2, \ldots, x_T\right)$).
- We wish to develop a generative model

$$p(x^T)$$

  that 'explains' the time series x^T.

- We cannot use the IID assumption $p(x^T) = \prod_t p(x_t)$. In general, we can use the chain rule (a.k.a. the general product rule)

$$\begin{aligned} p(x^T) &= p(x_T | x^{T-1}) p(x^{T-1}) \\ &= p(x_T | x^{T-1}) p(x_{T-1} | x^{T-2}) \cdots p(x_2 | x_1) p(x_1) \\ &= p(x_1) \prod_{t=2}^{T} p(x_t \mid x^{t-1}) \end{aligned}$$

- Generally, we will want to limit the depth of dependencies on previous observations. For example, a Kth-order linear Auto-Regressive (AR) model is given by

$$p(x_t \mid x^{t-1}) = \left(x_t \,\middle|\, \sum_{k=1}^{K} a_k x_{t-k}, \sigma^2\right)$$

  limits the dependencies to the past K samples.

State-space Models

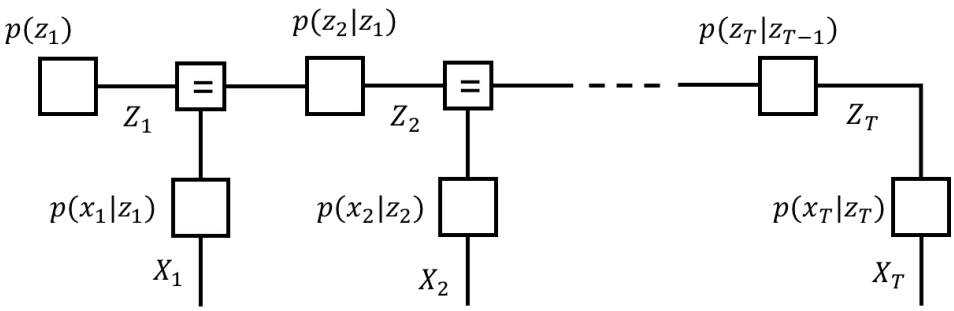
- A limitation of AR models is that they need a lot of parameters in order to create a flexible model. E.g., if x_t is an M-dimensional discrete variable, then a Kth-order AR model will have about M^K parameters.
- Similar to our work on Gaussian Mixture models, we can create a flexible dynamic system by introducing latent (unobserved) variables $z^T \triangleq \left(z_1, z_2, \ldots, z_T\right)$ (one z_t for each observation x_t). In dynamic systems, z_t are called state variables.

- A state space model is a particular latent variable dynamical model defined by

$$p(x^T, z^T) = p(z_1) \underbrace{\prod_{t=2}^{T} p(z_t \mid z_{t-1})}_{\text{state transitions}} \underbrace{\prod_{t=1}^{T} p(x_t \mid z_t)}_{\text{observations}}$$

  - The condition $p(z_t \mid z^{t-1}) = p(z_t \mid z_{t-1})$ is called a 1st-order Markov condition.

- The Forney-style factor graph for a state-space model:



Hidden Markov Models and Linear Dynamical Systems

- A Hidden Markov Model (HMM) is a specific state-space model with discrete-valued state variables z_t .

- Typically, z_t is a K-dimensional one-hot coded latent 'class indicator' with transition probabilities $a_{jk} \triangleq p(z_{tk} = 1 \mid z_{t-1,j} = 1)$, or equivalently,

$$p(z_t \mid z_{t-1}) = \prod_{k=1}^{K} \prod_{j=1}^{K} a_{jk}^{z_{t-1,j} \cdot z_{tk}}$$

  which is usually accompanied by an initial state distribution $p(z_{1k} = 1) = \pi_k$.

- The classical HMM has also discrete-valued observations but in pratice any (probabilistic) observation model $p(x_t \mid z_t)$ may be coupled to the hidden Markov chain.

- Another well-known state-space model with continuous-valued state variables z_t is the (Linear) Gaussian Dynamical System (LGDS), which is defined as

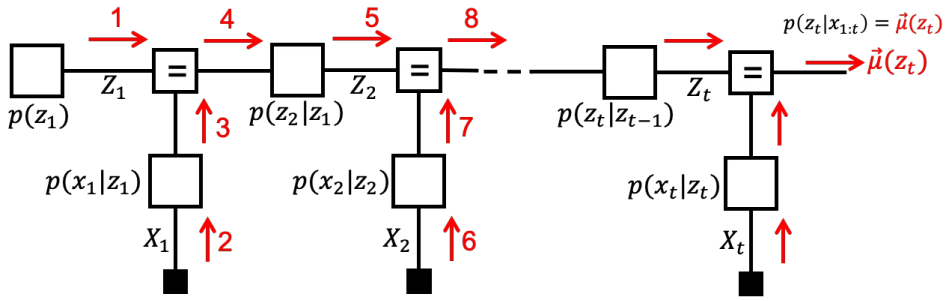
$$\begin{aligned} p(z_t \mid z_{t-1}) &= \left(Az_{t-1}, \Sigma_z\right) \\ p(x_t \mid z_t) &= \left(Cz_t, \Sigma_x\right) \\ p(z_1) &= \left(\mu_1, \Sigma_1\right) \end{aligned}$$

- Note that the joint distribution over all states and observations $\{(x_1, z_1), \ldots, (x_t, z_t)\}$ is a (large-dimensional) Gaussian distribution. This means that, in principle, every inference problem on the LGDS model also leads to a Gaussian distribution.

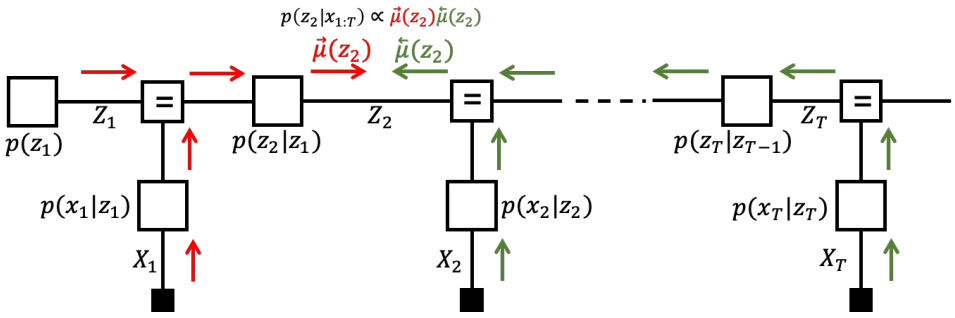
- HMM's and LGDS's (and variants thereof) are at the basis of a wide range of complex information processing systems, such as speech and language recognition, robotics and automatic car navigation, and even processing of DNA sequences.

Common Signal Processing Tasks as Message Passing-based Inference

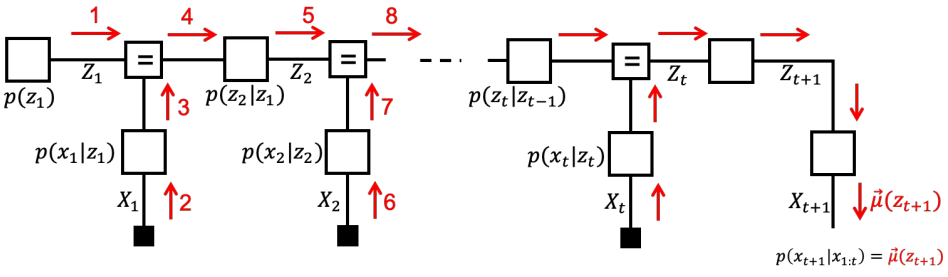
- As we have seen, inference tasks in linear Gaussian state space models can be analytically solved.
- However, these derivations quickly become cumbersome and prone to errors.
- Alternatively, we could specify the generative model in a (Forney-style) factor graph and use automated message passing to infer the posterior over the hidden variables. Here follows some examples.
- Filtering, a.k.a. state estimation: estimation of a state (at time step t), based on past and current (at t) observations.



- Smoothing: estimation of a state based on both past and future observations. Needs backward messages from the future.



- Prediction: estimation of future state or observation based only on observations of the past.



Kalman Filtering

- Technically, a Kalman filter is the solution to the recursive estimation (inference) of the hidden state z_t based on past observations in an LGDS, i.e., Kalman filtering solves the problem $p(z_t \mid x^t)$ based on the previous estimate $p(z_{t-1} \mid x^{t-1})$ and a new observation x_t (in the context of the given model specification of course).

- Let's infer the Kalman filter for a scalar linear Gaussian dynamical system:

$$p(z_t \mid z_{t-1}) = \quad (z_t \mid az_{t-1}, \sigma_z^2)$$
$$p(x_t \mid z_t) = \quad (x_t \mid cz_t, \sigma_x^2)$$

- Kalman filtering comprises inferring $p(z_t \mid x^t)$ from a given prior estimate $p(z_{t-1} \mid x^{t-1})$ (available after the previous time step) and a new observation x_t. Let us assume that

$$p(z_{t-1} \mid x^{t-1}) = \quad (z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)$$

- Note that everything is Gaussian, so it is in principle possible to execute inference problems analytically and the result will be a Gaussian posterior:
  - (In the following derivation we make use of the renormalization equality

$$(x \mid cz, \sigma^2) = \frac{1}{c} \left(z \mid \frac{x}{c}, \left(\frac{\sigma}{c}\right)^2\right).$$

$$p(z_t \mid x^t) = p(z_t \mid x_t, x^{t-1}) \propto p(x_t, z_t \mid x^{t-1})$$

posterior

$$\propto p(x_t \mid z_t) p(z_t \mid x^{t-1})$$

$$= p(x_t \mid z_t) \sum_{z_{t-1}} p(z_t, z_{t-1} \mid x^{t-1})$$

$$= p(x_t \mid z_t) \sum_{z_{t-1}} p(z_t \mid z_{t-1})\ p(z_{t-1} \mid x^{t-1})$$

observation state transition prior

$$= (x_t \mid cz_t, \sigma_x^2) \sum_{z_{t-1}} (z_t \mid az_{t-1}, \sigma_z^2)\ \ (z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)$$

$$= \frac{1}{c} \left(z_t \mid \frac{x_t}{c}, \left(\frac{\sigma_x}{c}\right)^2\right) \sum_{z_{t-1}} \frac{1}{a} \left(z_{t-1} \mid \frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2\right)\ \ (z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)$$

use Gaussian multiplication formula SRG-6

$$\propto \left(z_t \mid \frac{x_t}{c}, \left(\frac{\sigma_x}{c}\right)^2\right) \cdot \left(z_t \mid a\mu_{t-1}, \sigma_z^2 + \left(a\sigma_{t-1}\right)^2\right)$$

use SRG-6 again

$$\propto \left(z_t \mid \mu_t, \sigma_t^2\right)$$

with

$$\rho_t^2 = a^2 \sigma_{t-1}^2 + \sigma_z^2$$

$$K_t = \frac{c\rho_t^2}{c^2 \rho_t^2 + \sigma_x^2}$$

$$\mu_t = a\mu_{t-1} + K_t \cdot \left(x_t - ca\mu_{t-1}\right)$$

prior prediction prediction error

$$\sigma_t^2 = \left(1 - c \cdot K_t\right)\rho_t^2$$

- Kalman filtering consists of computing/updating these last four equations for each new observation (x_t). This is a very efficient recursive algorithm to estimate the state z_t from all observations (until t).
- It turns out that it's also possible to get an analytical result for $p(x_t \mid x^{t-1})$, which is the model evidence in a filtering context. See optional slides for details.

Multi-dimensional Kalman Filtering

- The Kalman filter equations can also be derived for multidimensional state-space models. In particular, for the model

$$z_t = Az_{t-1} + \quad (0, \Gamma)$$
$$x_t = Cz_t + \quad (0, \Sigma)$$

the Kalman filter update equations for the posterior $p(z_t | x^t) = \quad \left(z_t | \mu_t, V_t\right)$ are given by (see Bishop, pg.639)

$$P_t = AV_{t-1}A^T + \Gamma$$
$$K_t = P_t C^T \cdot \left(CP_t C^T + \Sigma\right)^{-1}$$
$$\mu_t = A\mu_{t-1} + K_t \cdot \left(x_t - CA\mu_{t-1}\right)$$
$$V_t = \left(I - K_t C\right)P_t$$

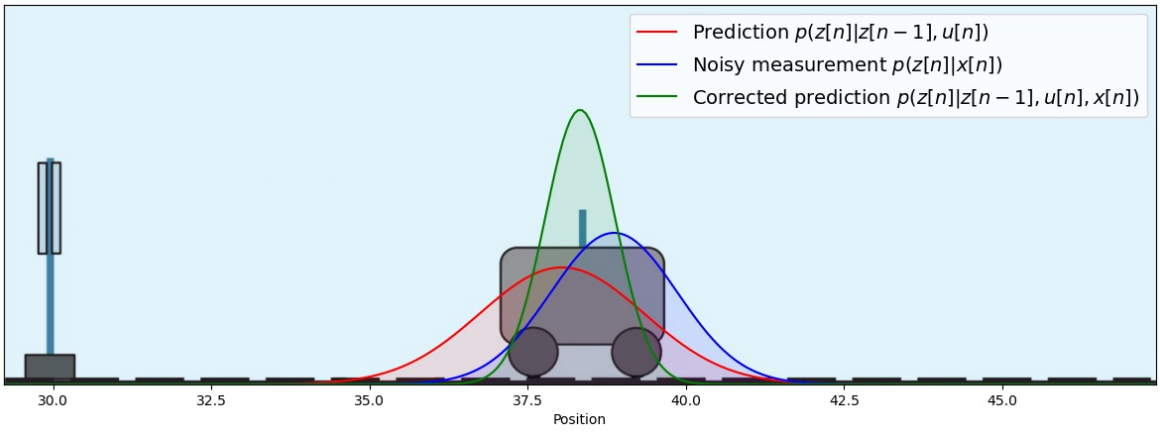
Kalman Filtering and the Cart Position Tracking Example Revisited

CODE EXAMPLE

- We can now solve the cart tracking problem of the introductory example by implementing the Kalman filter.

```
using Pkg;Pkg.activate("probprog/workspace/");Pkg.instantiate()
IJulia.clear_output();
```

```
using LinearAlgebra, PyPlot
include("scripts/cart_tracking_helpers.jl")

# Specify the model parameters
Δt = 1.0                    # assume the time steps to be equal in size
A = [1.0 Δt;
     0.0 1.0]
b = [0.5*Δt^2; Δt]
Σz = convert(Matrix,Diagonal([0.2*Δt; 0.1*Δt])) # process noise covariance
Σx = convert(Matrix,Diagonal([1.0; 2.0]))     # observation noise covariance;

# Generate noisy observations
n = 10                # perform 10 timesteps
z_start = [10.0; 2.0] # initial state
u = 0.2 * ones(n)     # constant input u
noisy_x = generateNoisyMeasurements(z_start, u, A, b, Σz, Σx);

m_z = noisy_x[1]                                    # initial predictive mean
V_z = A * (1e8*Diagonal(I,2) * A') + Σz             # initial predictive covariance

for t = 2:n
    global m_z, V_z, m_pred_z, V_pred_z
    #predict
    m_pred_z = A * m_z + b * u[t]                   # predictive mean
    V_pred_z = A * V_z * A' + Σz                    # predictive covariance
    #update
    gain = V_pred_z * inv(V_pred_z + Σx)            # Kalman gain
    m_z = m_pred_z + gain * (noisy_x[t] - m_pred_z) # posterior mean update
    V_z = (Diagonal(I,2)-gain)*V_pred_z             # posterior covariance update
end
println("Prediction: ",ProbabilityDistribution(Multivariate,GaussianMeanVariance,m=m_pred_z,v=V_pr
ed_z))
println("Measurement: ", ProbabilityDistribution(Multivariate,GaussianMeanVariance,m=noisy_x[n],v=
Σx))
println("Posterior: ", ProbabilityDistribution(Multivariate,GaussianMeanVariance,m=m_z,v=V_z))
plotCartPrediction2(m_pred_z[1], V_pred_z[1], m_z[1], V_z[1], noisy_x[n][1], Σx[1][1]);
```



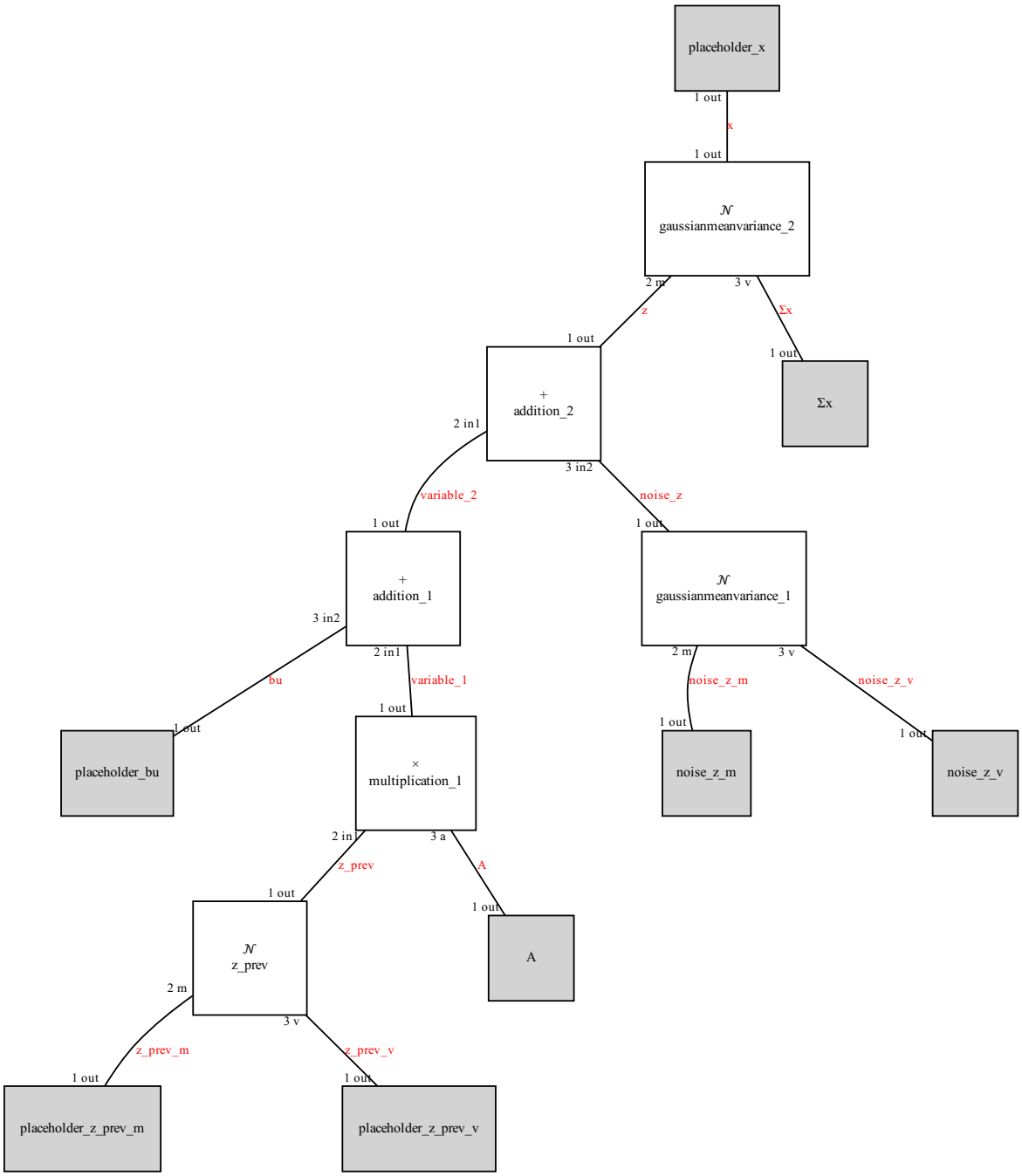

```
Prediction: 𝒩(m=[38.04, 3.24], v=[[1.30, 0.39][0.39, 0.34]])

Measurement: 𝒩(m=[38.87, 1.01], v=[[1.00, 0.00][0.00, 2.00]])

Posterior: 𝒩(m=[38.33, 3.09], v=[[0.55, 0.15][0.15, 0.24]])
```

The Cart Tracking Problem Revisited: Inference by Message Passing

- Let's solve the cart tracking problem by sum-product message passing in a factor graph like the one depicted above. All we have to do is create factor nodes for the state-transition model $p(z_t|z_{t-1})$ and the observation model $p(x_t|z_t)$. Then we just build the factor graph and let ForneyLab execute the message passing schedule.

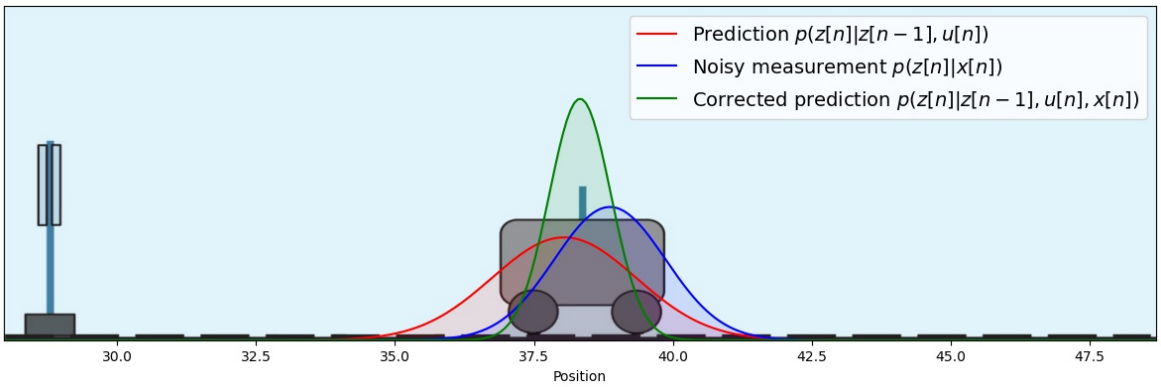
- Since the factor graph is just a concatination of n identical "sections" (one for each time step), we only have to specify a single section. When running the message passing algorithm we will explictly use the posterior of the previous timestep as prior in the next one. Let's build a section of the factor graph:

```
fg = FactorGraph()
z_prev_m = Variable(id=:z_prev_m); placeholder(z_prev_m, :z_prev_m, dims=(2,))
z_prev_v = Variable(id=:z_prev_v); placeholder(z_prev_v, :z_prev_v, dims=(2,2))
bu = Variable(id=:bu); placeholder(bu, :bu, dims=(2,))

@RV z_prev ~ GaussianMeanVariance(z_prev_m, z_prev_v, id=:z_prev) # p(z_prev)
@RV noise_z ~ GaussianMeanVariance(constant(zeros(2), id=:noise_z_m), constant(Σz, id=:noise_z_v))
 # process noise
@RV z = constant(A, id=:A) * z_prev + bu + noise_z; z.id = :z # p(z|z_prev)  (state transition mode
l)
@RV x ~ GaussianMeanVariance(z, constant(Σx, id=:Σx)) # p(x|z)  (observation model)
placeholder(x, :x, dims=(2,));
ForneyLab.draw(fg)
```



Now that we've built the factor graph, we can perform Kalman filtering by inserting measurement data into the factor graph and performing message passing.

```
include("scripts/cart_tracking_helpers.jl")
algo = messagePassingAlgorithm(z)
source_code = algorithmSourceCode(algo)
eval(Meta.parse(source_code))
marginals = Dict()
messages = Array{Message}(undef,6)
z_prev_m_0 = noisy_x[1]
z_prev_v_0 = A * (1e8*Diagonal(I,2) * A') + Σz
for t=2:n
    data = Dict(:x => noisy_x[t], :bu => b*u[t],:z_prev_m => z_prev_m_0, :z_prev_v => z_prev_v_0)
    step!(data, marginals, messages) # perform msg passing (single timestep)
    # Posterior of z becomes prior of z in the next timestep:
    z_prev_m_0 = ForneyLab.unsafeMean(marginals[:z])
    z_prev_v_0 = ForneyLab.unsafeCov(marginals[:z])
end
# Collect prediction p(z[n]|z[n-1]), measurement p(z[n]|x[n]), corrected prediction p(z[n]|z[n-1],
x[n])
prediction      = messages[5].dist # the message index is found by manual inspection of the schedu
le
measurement     = messages[6].dist
corr_prediction = convert(ProbabilityDistribution{Multivariate, GaussianMeanVariance}, marginals[:
z])
println("Prediction: ",prediction)
println("Measurement: ",measurement)
println("Posterior: ", corr_prediction)

# Make a fancy plot of the prediction, noisy measurement, and corrected prediction after n timeste
ps
plotCartPrediction(prediction, measurement, corr_prediction);
```




```
Prediction: 𝒩(m=[38.04, 3.24], v=[[1.30, 0.39][0.39, 0.34]])

Measurement: 𝒩(m=[38.87, 1.01], v=[[1.00, 0.00][0.00, 2.00]])

Posterior: 𝒩(m=[38.33, 3.09], v=[[0.55, 0.15][0.15, 0.24]])
```

- Note that both the analytical Kalman filtering solution and the message passing solution lead to the same results. The advantage of message passing-based inference with ForneyLab is that we did not need to derive any inference equations. ForneyLab took care of all that.

Recap Dynamical Models

- Dynamical systems do not obey the sample-by-sample independence assumption, but still can be specified, and state and parameter estimation equations can be solved by similar tools as for static models.

- Two of the more famous and powerful models with latent states include the hidden Markov model (with discrete states) and the Linear Gaussian dynamical system (with continuous states).

- For the LGDS, the Kalman filter is a well-known recursive state estimation procedure. The Kalman filter can be derived through Bayesian update rules on Gaussian distributions.

- If anything changes in the model, e.g., the state noise is not Gaussian, then you have to re-derive the inference equations again from scratch and it may not lead to an analytically pleasing answer.

- ⇒ Generally, we will want to automate the inference processes. As we discussed, message passing in a factor graph is a visually appealing method to automate inference processes. We showed how Kalman filtering emerged naturally by automated message passing.

OPTIONAL SLIDES

Proof of Kalman filtering equations including evidence updating

- Now let's proof the Kalman filtering equations including evidence updating by probabilistic calculus:

$$
\begin{aligned}
\underset{\text{posterior}}{p(z_t \mid x^t)} &= p(z_t \mid x_t, x^{t-1}) \\
&= \frac{p(x_t, z_t \mid x^{t-1})}{p(x_t \mid x^{t-1})} \\
&= \frac{p(x_t \mid z_t) p(z_t \mid x^{t-1})}{p(x_t \mid x^{t-1})} \\
&= \frac{p(x_t \mid z_t) \sum_{z_{t-1}} p(z_t, z_{t-1} \mid x^{t-1})}{p(x_t \mid x^{t-1})} \\
&= \frac{\underset{\text{likelihood}}{p(x_t \mid z_t)} \sum_{z_{t-1}} \underset{\text{state transition}}{p(z_t \mid z_{t-1})} \underset{\text{prior}}{p(z_{t-1} \mid x^{t-1})}}{p(x_t \mid x^{t-1})} \\
&= \frac{\mathcal{N}(x_t \mid c z_t, \sigma_x^2) \sum_{z_{t-1}} \mathcal{N}(z_t \mid a z_{t-1}, \sigma_z^2) \; \mathcal{N}(z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)}{\underset{\text{evidence}}{p(x_t \mid x^{t-1})}}
\end{aligned}
$$

- The posterior $p(z_t \mid x^t)$ is proportional to the numerator, which by making use of the renormalization equality

$$
\mathcal{N}(x \mid cz, \sigma^2) = \frac{1}{c} \cdot \mathcal{N}\left(z \mid \frac{x}{c}, \left(\frac{\sigma}{c}\right)^2\right),
$$

  can be evaluated with Gaussian multiplication rules:

$$(x_t|\, cz_t, \sigma_x^2) \sum_{z_{t-1}} (z_t \mid az_{t-1}, \sigma_z^2)\ \ (z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)$$

use renormalization

$$= \ (x_t|\, cz_t, \sigma_x^2) \sum_{z_{t-1}} \frac{1}{a}\ \ \left(z_{t-1}\,|\,\frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2\right)\ \ (z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2)$$

use Gaussian multiplication formula SRG-6

$$= \frac{1}{a}\ \ (x_t|\, cz_t, \sigma_x^2) \sum_{z_{t-1}} \left(\mu_{t-1}\,|\,\frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2 + \sigma_{t-1}^2\right)\ \ (z_{t-1}\ |\ \cdot\ ,\ \cdot\ )$$

sums to 1

not a function of z_{t-1}

$$= \frac{1}{a}\ \ (x_t|\, cz_t, \sigma_x^2) \qquad \left(\mu_{t-1}\,|\,\frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2 + \sigma_{t-1}^2\right)$$

use renormalization rule

use renormalization rule

$$= \frac{1}{c}\ \ \left(z_t\,|\,\frac{x_t}{c}, \left(\frac{\sigma_x}{c}\right)^2\right)\ \ \left(z_t \mid a\mu_{t-1}, \sigma_z^2 + \left(a\sigma_{t-1}\right)^2\right)$$

use SRG-6 again

$$= \frac{1}{c}\ \ \left(\frac{x_t}{c}\,|a\mu_{t-1}, \left(\frac{\sigma_x}{c}\right)^2 + \sigma_z^2 + \left(a\sigma_{t-1}\right)^2\right)\ \ \left(z_t \mid \mu_t, \sigma_t^2\right)$$

use renormalization

$$= \ \left(x_t \mid ca\mu_{t-1}, \sigma_x^2 + c^2(\sigma_z^2 + a^2\sigma_{t-1}^2)\right)\ \ \left(z_t \mid \mu_t, \sigma_t^2\right)$$

evidence $p(x_t|x^{t-1})$ posterior $p(z_t|x^t)$

with

$$\rho_t^2 = a^2\sigma_{t-1}^2 + \sigma_z^2$$

$$K_t = \frac{c\rho_t^2}{c^2\rho_t^2 + \sigma_x^2}$$

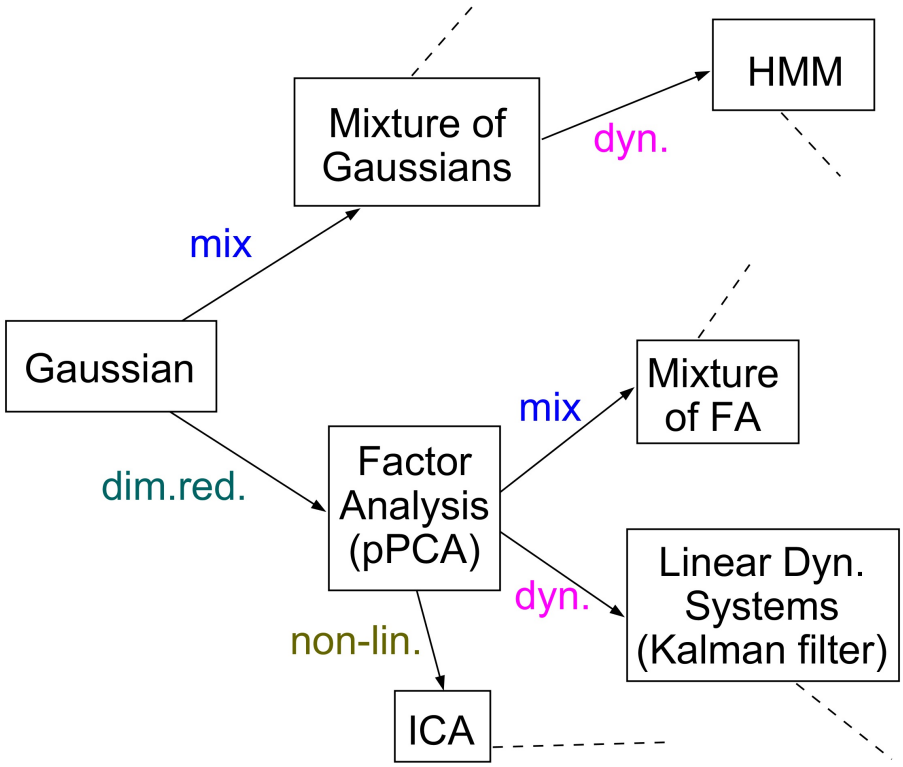
$$\mu_t = a\mu_{t-1} \qquad + K_t \cdot \left(x_t - ca\mu_{t-1}\right)$$

prior prediction prediction error

$$\sigma_t^2 = \left(1 - c \cdot K_t\right)\rho_t^2$$

Extensions of Generative Gaussian Models

- Using the methods of the previous lessons, it is possible to create your own new models based on stacking Gaussian and categorical distributions in new ways:



Intelligent Agents and Active Inference

Preliminaries

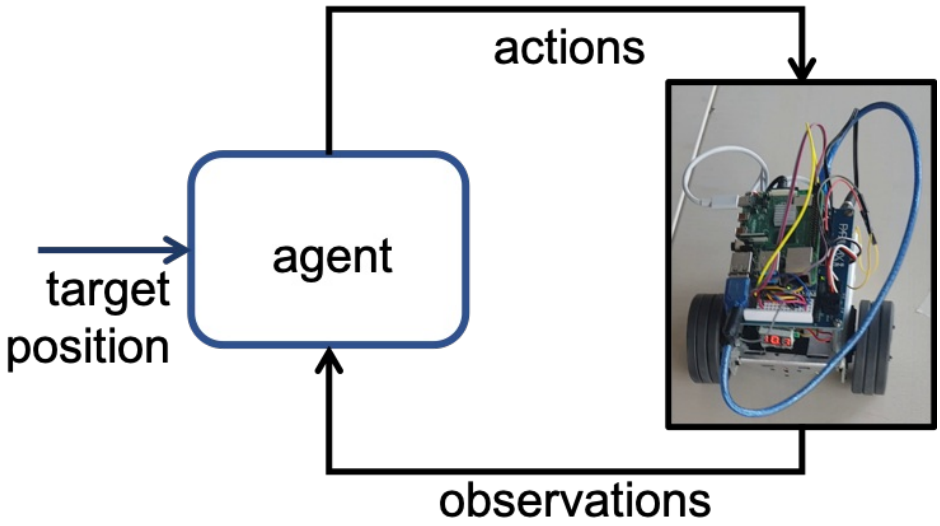
- Goal
  - Introduction to Active Inference and application to the design of synthetic intelligent agents
- Materials
  - Mandatory
    - These lecture notes
    - Karl Friston – 2016 – The Free Energy Principle (video)
  - Optional
    - Raviv (2018), The Genius Neuroscientist Who Might Hold the Key to True AI.
      - Interesting article on Karl Friston, who is a leading theoretical neuroscientist working on a theory that relates life and intelligent behavior to physics (and Free Energy minimization). (highly recommended)
    - Kirsch (2019), Theories of Intelligence: Active Inference
      - A nice tutural blog on active inference.
    - Van de Laar and De Vries (2019), Simulating Active Inference Processes by Message Passing
      - How to implement active inference by message passing in a Forney-style factor graph.
  - References
    - Friston (2013), Life as we know it
    - Conant and Ashby (1970), Every good regulator of a system must be a model of that system

Agents

- In the previous lessons we assumed that a data set was given.
- In this lesson we consider agents. An agent is a system that interacts with its environment through both sensors and actuators.
- Crucially, by acting onto the environment, the agent is able to affect the data that it will sense in the future.
    - As an example, by changing the direction where I look, I can affect the sensory data that will be sensed by my retina.
- With this definition of an agent, (biological) organisms are agents, and so are robots, self-driving cars, etc.
- In an engineering context, we are particularly interesting in agents that behave with a purpose (with a goal in mind), e.g., to drive a car or to design a speech recognition algorithm.
- In this lesson, we will describe how goal-directed behavior by biological (and synthetic) agents can also be interpreted as minimization of a free energy functional $F[q]$.

Illustrative Example: Steering a cart to a parking spot

- In this example, we consider a cart that can move in a 1D space. At each time step the cart can be steered a bit to the left or right by a controller (the "agent"). The agent's knowledge about the cart's process dynamics (equations of motion) are known up to some additive Gaussian process noise. The agent also makes noisy observations of the position and velocity of the cart. Your challenge is to design an agent that steers the car to the zero position. (The agent should be specified as a probabilistic model and the control signal should be formulated as a Bayesian inference task).



- Solution at the end of this lesson.

Karl Friston and the Free Energy Principle

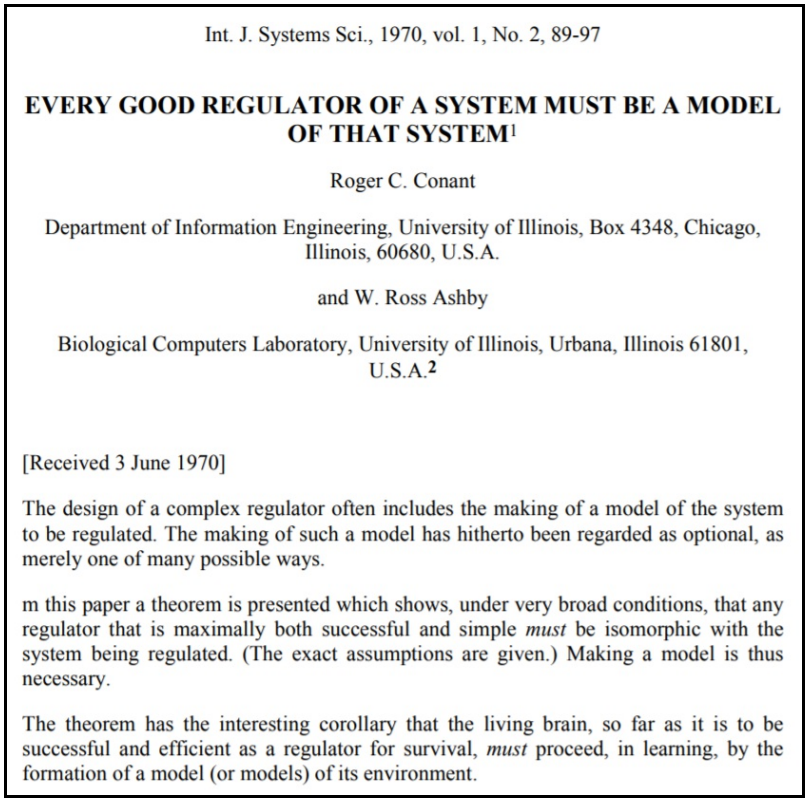
- We begin with a motivating example that requires "intelligent" goal-directed decision making: assume that you are an owl and that you're hungry. What are you going to do?
- Have a look at Prof. Karl Friston's answer in this video segment by on the cost function for intelligent behavior. (Do watch the video!)
- Friston argues that intelligent decision making (behavior, action making) by an agent requires minimization of a functional of beliefs.
- Friston further argues (later in the lecture and his papers) that this functional is a (variational) free energy (to be defined below), thus linking decision making to Bayesian inference.
- In fact, Friston's Free Energy Principle (FEP) claims that all biological self-organizing processes (including brain processes) can be described as Free Energy minimization in a probabilistic model.
    - This includes perception, learning, attention mechanisms, recall, action and decision making, etc.
- Taking inspiration from FEP, if we want to develop synthetic "intelligent" agents, we have (only) two issues to consider:

    1. The specification of the FE functional (includes specification of generative model and constraints on the approximate posterior, a.k.a. the "recognition" model).
    2. How to minimize the FE functional?

What Makes a Good Agent?

- What should the agent's model be modeling? This question was (already) answered by Conant and Ashby (1970) as the good regulator theorem: every good regulator of a system must be a model of that system.
- From Conant and Ashby's paper (this statement was later finessed by Friston (2013)):

> The theory has the interesting corollary that the living brain, insofar as it is successful and efficient as a regulator for survival, must proceed, in learning, by the formation of a model (or models) of its environment."



Int. J. Systems Sci., 1970, vol. 1, No. 2, 89-97

EVERY GOOD REGULATOR OF A SYSTEM MUST BE A MODEL OF THAT SYSTEM[1]

Roger C. Conant

Department of Information Engineering, University of Illinois, Box 4348, Chicago, Illinois, 60680, U.S.A.

and W. Ross Ashby

Biological Computers Laboratory, University of Illinois, Urbana, Illinois 61801, U.S.A.[2]

[Received 3 June 1970]

The design of a complex regulator often includes the making of a model of the system to be regulated. The making of such a model has hitherto been regarded as optional, as merely one of many possible ways.

m this paper a theorem is presented which shows, under very broad conditions, that any regulator that is maximally both successful and simple *must* be isomorphic with the system being regulated. (The exact assumptions are given.) Making a model is thus necessary.

The theorem has the interesting corollary that the living brain, so far as it is to be successful and efficient as a regulator for survival, *must* proceed, in learning, by the formation of a model (or models) of its environment.

Active Inference Agents

- We will follow the idea that an agent needs to hold a generative model for its environment, which is observed through sensory channels. The environmental dynamics can be affected through actions onto the environment.
- Agents that follow the FEP and infer actions by inference in a generative model of the environment are engaged in a process called active inference.

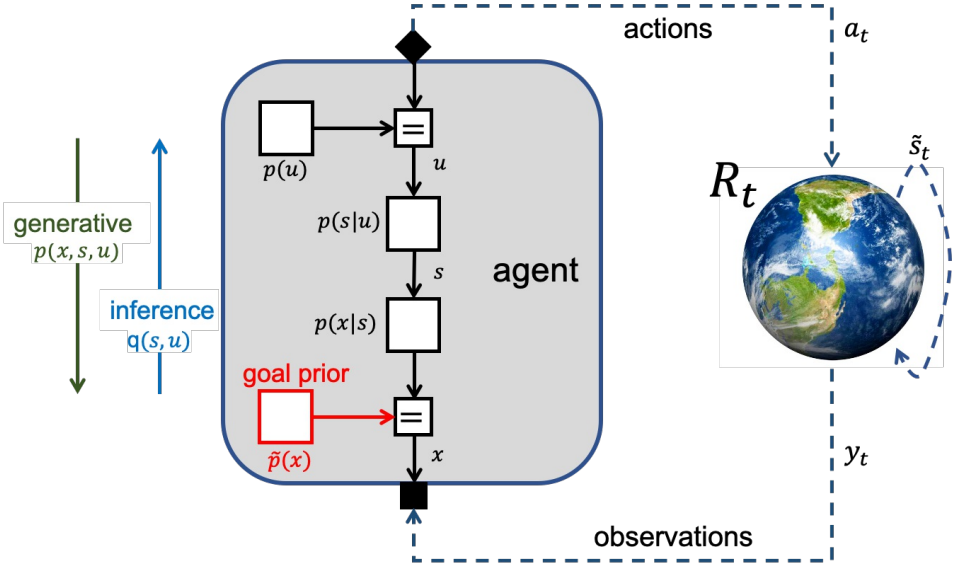
- Technically, an active inference-based agent comprises:

1. A free energy functional $F[q] = \mathrm{E}_q\left[\log\frac{q(z)}{p(x,z)}\right]$, where
   - $p(x,z) = \prod_k p(x_k, z_k \mid z_{k-1})$ is a generative model with observations $\{x_k\}$, latent variables $\{z_k\} = \left\{\{s_k\}, \{u_k\}, \{\theta_k\}\right\}$ and k is a time index.

- $q(z)$ is a recognition model.

2. A recipe to minimize the free energy $F[q]$

- Let's draw a diagram to show the interactions between an active inference agent and its environment.



- In the model above, the hidden variables $\{z_k\}$ of the agent comprise internal states $\{s_k\}$, control variables $\{u_k\}$ (which are "observed" by the environment as actions $\{a_k\}$), and parameters $\{\theta_k\}$.
- In neuroscience/psychology parlance,
  - behavior (movement) is inference for the control signals (u)
  - perception is inference for the internal states (s).
  - learning is inference for the parameters (θ)
- We also assume that the agent interacts with an environment, which we represent by a dynamic model

$$(y_t, \tilde{s}_t) = R_t\left(a_t, \tilde{s}_{t-1}\right)$$

where a_t are actions , y_t are outcomes and $\tilde{s}_t$ holds the environmental states.

- In the above equations, u_t and x_t are owned by the agent model, whereas a_t and y_t are variables in the environment model.
- The agent can push actions a_t onto the environment and measure responses y_t, but has no access to the environmental states $\tilde{s}_t$.
- Interactions between the agent and environment are described by

$$a_t \sim q(u_t)$$
$$x_t = y_t$$

iow, actions are drawn from the posterior over control signals.

- Note that this system implies a recursive dependency since the agent's future observations depend on the agent's current (and past) actions:

$$x_{t+1} = x_{t+1}\left(a_{t+1}\right) = x_{t+1}\left(a_{t+1}\left(u_{t+1}\left(x_t\left(a_t(\cdots)\right)\right)\right)\right)$$

  - ⇒ As a result, the agent actively engages in selecting its own data set!

Goal-directed Behavior

- Biological agents select their observations by controling their environment. Perception (and learning) serve to improve this data selection process by updating beliefs about the state of the world.
- This process begs the question: if a (biological) agent seeks out observations, then which observations is the agent interested in? I.o.w., does the agent have a goal "in mind" when it engages in active data selection?
- Yes! Agents set preferences for future observations by setting prior distributions on future observations!
  - E.g., a self-driving agent in a car expects to observe no collisions.
- Thus, the generative model for an active inference agent at time t includes variables at future time steps and can be run forward to make predictions (beliefs) about future observations $x_{t+1:T}$.

Active Inference Agent Model specification

- We assume that agents live in a dynamic environment and consider the following generative model for the agent (omitting parameters θ), and assuming the current time is t:

$$p'(x,s,u) = p(s_{t-1}) \underbrace{\prod_{k=t}^{t+T} p(x_k \mid s_k) \cdot p(s_k \mid s_{k-1}, u_k)}_{\text{internal dynamics}} \cdot \underbrace{p(u_k)}_{\substack{\text{control} \\ \text{prior}}}$$

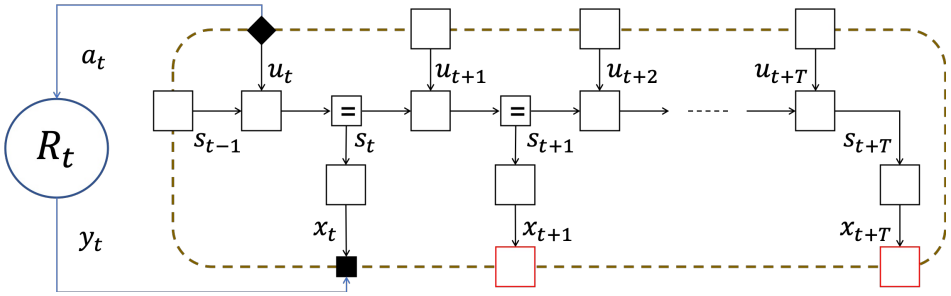
  - Note that the generative model includes future time steps.
- In order to infer goal-driven (i.e., purposeful) behavior, we now add prior beliefs $\tilde{p}(x)$ about desired future observations, leading to an extended agent model:

$$\begin{aligned} p(x,s,u) &= \frac{p'(x,s,u)\tilde{p}(x)}{\int_x p'(x,s,u)\tilde{p}(x)\mathrm{d}x} \\ &\propto \underbrace{p(s_{t-1}) \prod_{k=t}^{t+T} p(x_k \mid s_k) p(s_k \mid s_{k-1}, u_k) p(u_k)}_{\text{original generative model}} \underbrace{\tilde{p}(x_k)}_{\substack{\text{extension} \\ \text{"goal prior"}}} \end{aligned}$$

  - $\tilde{p}(x)$ encodes priors beliefs by the agent about future observations.
- Goal-directed behavior follows from inference for controls (actions) at t, based on expectations (encoded by priors) about future ($> t$) observations.
- ⇒ Actions fulfill expectations about the future!

FFG for Active Inference Agent Model

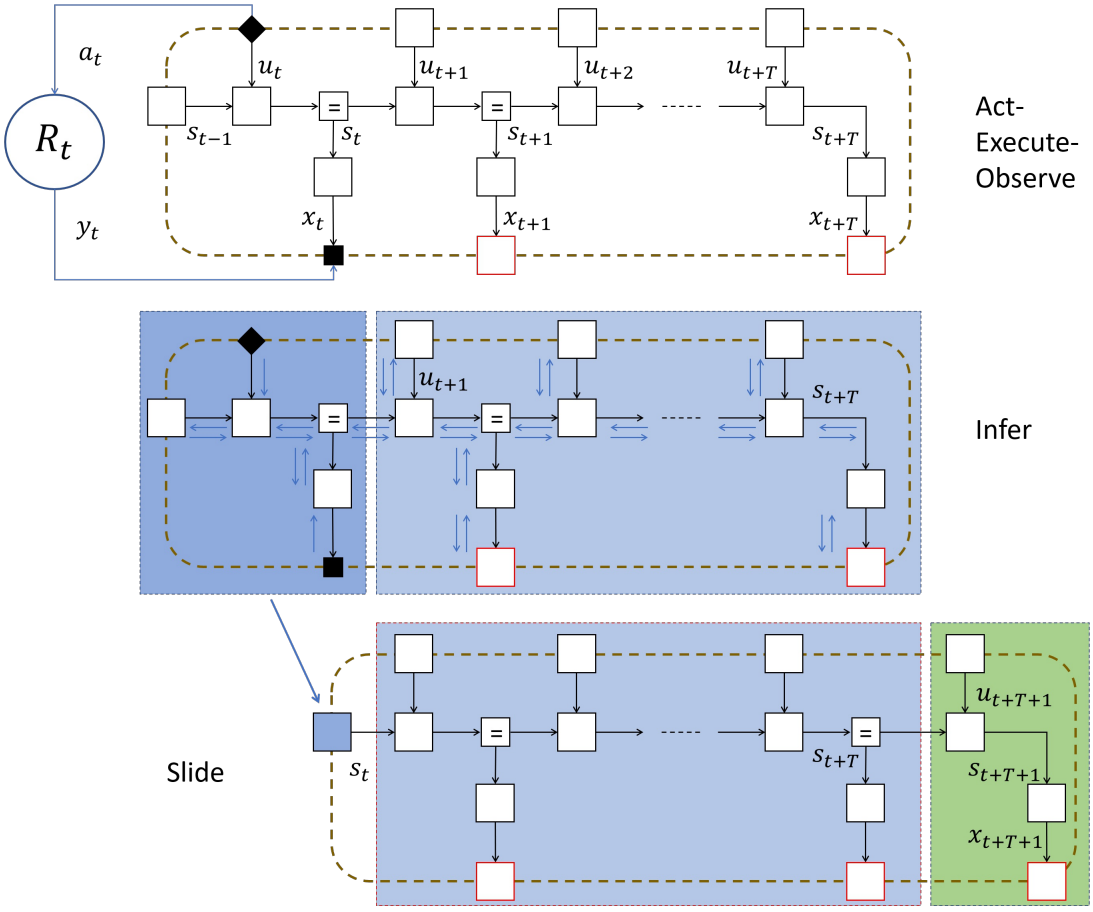
- After selecting an action a_t and making an observation y_t, the FFG for the extended generative model is given by the following FFG:



- The (brown) dashed box is the agent's Markov blanket. Given the states on the Markov blanket, the internal states of the agent are independent of the state of the world.

How to minimize FE: Online Active Inference

- Online active inference proceeds by iteratively executing three stages: (1) act-execute-observe, (2) infer the next control/action, (3) slide forward

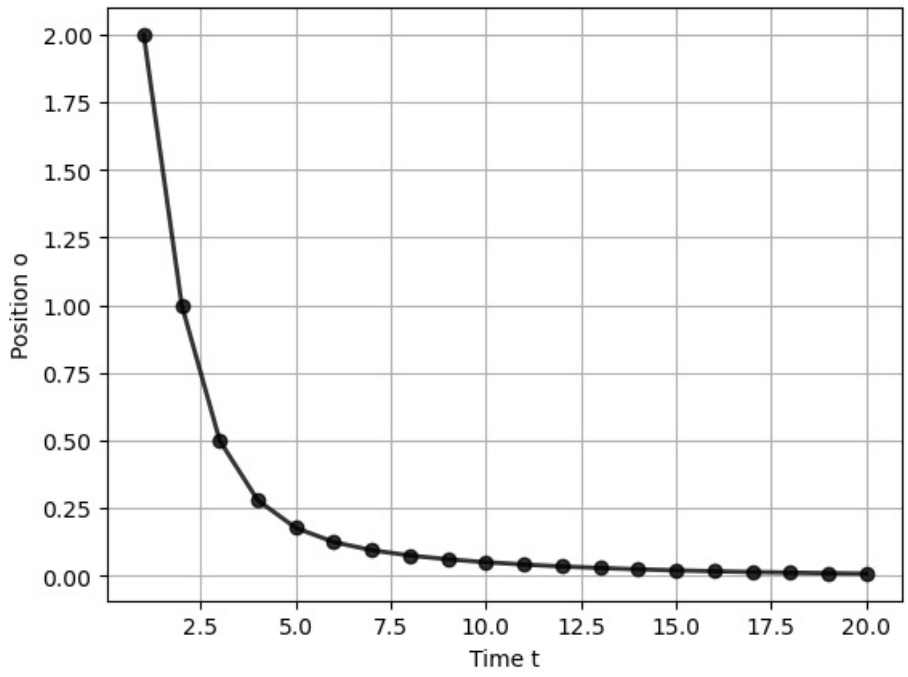



```
using Pkg;Pkg.activate("probprog/workspace");Pkg.instantiate()
IJulia.clear_output();
```

The Cart Park Problem Revisited

Here we solve the cart parking problem as stated at the beginning of this lesson. We first specify a generative model for the agent's environment (which is the observed noisy position of the cart) and then constrain future observations by a prior distribution that is located on the target parking spot. Next, we schedule a message passing-based inference algorithm for the next action. This is followed by executing the "Act-execute-observe --> infer --> slide" procedure to infer a sequence of consecutive actions. Finally, the position of the cart over time is plotted. Note that the cart convergees onto the target spot.

```
using PyPlot, LinearAlgebra, ForneyLab
# Load helper functions. Feel free to explore these
include("ai_agent/environment_1d.jl")
include("ai_agent/helpers_1d.jl")
include("ai_agent/agent_1d.jl")

# Internal model perameters
gamma   = 100.0 # Transition precision
phi     = 10.0 # Observation precision
upsilon = 1.0 # Control prior variance
sigma   = 1.0 # Goal prior variance

T = 10 # Lookahead

# Build internal model
fg = FactorGraph()

o = Vector{Variable}(undef, T) # Observed states
s = Vector{Variable}(undef, T) # internal states
u = Vector{Variable}(undef, T) # Control states

@RV s_t_min ~ GaussianMeanVariance(placeholder(:m_s_t_min),
                                   placeholder(:v_s_t_min)) # Prior  state
u_t = placeholder(:u_t)
@RV u[1] ~ GaussianMeanVariance(u_t, tiny)
@RV s[1] ~ GaussianMeanPrecision(s_t_min + u[1], gamma)
@RV o[1] ~ GaussianMeanPrecision(s[1], phi)
placeholder(o[1], :o_t)

s_k_min = s[1]
for k=2:T
    @RV u[k] ~ GaussianMeanVariance(0.0, upsilon) # Control prior
    @RV s[k] ~ GaussianMeanPrecision(s_k_min + u[k], gamma) # State transition model
    @RV o[k] ~ GaussianMeanPrecision(s[k], phi) # Observation model
    GaussianMeanVariance(o[k],
                         placeholder(:m_o, var_id=:m_o_*k, index=k-1),
                         placeholder(:v_o, var_id=:v_o_*k, index=k-1)) # Goal prior
    s_k_min = s[k]
end

# Schedule message passing algorithm
algo = messagePassingAlgorithm(u[2]) # Infer internal states
source_code = algorithmSourceCode(algo)
eval(Meta.parse(source_code)) # Loads the step!() function for inference

s_0 = 2.0 # Initial State

N = 20 # Total simulation time

(execute, observe)  = initializeWorld() # Let there be a world
(infer, act, slide) = initializeAgent() # Let there be an agent

# Step through action-perception loop
u_hat = Vector{Float64}(undef, N) # Actions
o_hat = Vector{Float64}(undef, N) # Observations
for t=1:N
    u_hat[t] = act() # Evoke an action from the agent
               execute(u_hat[t]) # The action influences hidden external states
    o_hat[t] = observe() # Observe the current environmental outcome (update p)
               infer(u_hat[t], o_hat[t]) # Infer beliefs from current model state (update q)
               slide() # Prepare for next iteration
end

# Plot active inference results
plotTrajectory(u_hat, o_hat)
;
```



Video

- See this video to verfiy that the robot will be steered to the correct parking spot even after an "adversarial" intervention :). (This project was completed by Burak Ergul as part of this MSc graduation project).

Extensions

- If interested, here is a link to a more detailed version of the 1D parking problem.
- We also have a 2D version of this cart parking problem implemented on Raspberry Pi-based robot. (Credits for this implemention to Thijs van de Laar and Burak Ergul).

OPTIONAL SLIDES

Specification of Free Energy

- Consider the agent's inference task at time step t, right after having selected an action a_t and having made an observation y_t.
- As usual, we record actions and observations by substituting the values into the generative model(in the Act-Execute-Observe phase):

$$p(x,s,u) \propto \underbrace{p(x_t = y_t \mid s_t)}_{\text{observation}} \underbrace{p(s_t \mid s_{t-1}, u_t)}_{\text{action}} p(s_{t-1}) p(u_t = a_t) \cdot \underbrace{\prod_{k=t+1}^{t+T} p(x_k \mid s_k) p(s_k \mid s_{k-1}, u_k) p(u_k) p^+(x_k)}_{\text{future}}$$

- Note that (future) x is also a latent variable and hence we include x in the recognition model.
- This leads to the following free energy functional

$$F[q] \propto \sum_{x,s,u} q(x,s,u) \log \frac{q(x,s,u)}{p(x,s,u)}$$

FE Decompositions

- Lots of interesting FE decompositions are possible again. For instance

$$\begin{aligned} F[q] &\propto \sum_{x,s,u} q(x,s,u) \log \frac{q(x,s,u)}{p(x,s,u)} \\ &= \sum_u q(u) \underbrace{\sum_{x,s} q(x,s \mid u) \log \frac{q(x,s \mid u)}{p(x,s \mid u)}}_{F_u[q]} + \underbrace{\sum_u q(u) \log \frac{q(u)}{p(u)}}_{\text{complexity}} \end{aligned}$$

  breaks the FE into a complexity term and a term $F_u[q]$ that is conditioned on the policy u.
- It can be shown (exercise) that the optimal posterior for the policy is now given by

$$q^*(u) \propto p(u) \exp\left(-F_u^*\right)$$

- Let's consider a break-up $x = (x_t, x_{>t})$ with $x_{>t} = (x_{t+1}, \ldots, x_{t+T})$ that recognizes the distinction between already observed and future data. Then

$$F_u[q] = \underbrace{-\log p(x_t)}_{\substack{-\log \text{ (evidence)} \\ \text{(surprise)}}} + \underbrace{\sum_{x,s} q(x_{>t}, s \mid u) \log \frac{q(x_{>t}, s \mid u)}{p(x_{>t}, s \mid u)}}_{\substack{\text{divergence} \\ \text{(inference costs)}}}.$$

- The inference costs (divergence term) can be further decomposed to

$$\underbrace{-\sum_x q(x_{>t}) \log p(x_{>t})}_{\substack{\text{expected surprise} \\ \text{(goal-directed, pragmatic costs)}}} + \underbrace{\sum_{x,s} q(x_{>t}, s \mid u) \log \frac{q(x_{>t}, s \mid u)}{p(s \mid x_{>t}, u)}}_{\text{epistemic costs}}$$

- Minimizing goal-directed costs selects actions that (expect to) fullfil the priors over future observations. Minimization of epistemic ("knowledge seeking") costs leads to actions that maximize information gain about the environmental dynamics. This can be seen by further decomposition of

the epistemic costs into

$$\sum_{x,s} q(x_{>t}, s \mid u) \log \frac{q(s \mid u)}{p(s \mid x_{>t}, u)} + \sum_{x,s} q(x_{>t}, s \mid u) \log q(x_{>t} \mid s, u)$$
$$\approx \underbrace{\sum_{x,s} q(x_{>t}, s \mid u) \log \frac{q(s \mid u)}{q(s \mid x_{>t}, u)}}_{-\text{mutual information}} \underbrace{- \mathrm{E}_{q(s \mid u)}\left[H\left[q(x_{>t} \mid s, u)\right]\right]}_{\text{ambiguity}}$$

where we used the approximation $q(s \mid x_{>t}, u) \approx p(s \mid x_{>t}, u)$ to illuminate the link to the mutual information.

- Minimizing FE leads (approximately) to mutual information maximization between internal states s and observations x. In other words, FEM leads to actions that aim to seek out observations that are maximally informative about the hidden causes of these observations.
- Ambiguous states have uncertain mappings to observations. Minimizing FE leads to actions that try to avoid ambiguous states.
- In short, if the generative model includes variables that represent (yet) unobserved future observations, then action selection by FEM leads to a very sophisticated behavioral strategy that is maximally consistent with
  - Bayesian notions of model complexity
  - evidence from past observations
  - goal-directed imperatives by priors on future observations
  - epistemic (knowledge seeking) value maximization, both in terms of MI maximization and avoidance of ambiguous states
- All these imperatives are simultaneously represented and automatically balanced against each other in a single time-varying cost function (Free Energy) that needs no tuning parameters.
- (Just to be sure, you don't need to memorize these derivations nor are you expected to derive them on-the-spot. We present these decompositions only to provide insight into the multitude of forces that underlie FEM-based action selection.)

Free energy distribution in FFG